Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali
Corso di Laurea Triennale in Informatica

Firma digitale di pacchetti VoIP su Symbian

Tesi di Laurea in Architettura degli Elaboratori

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Presentata da:
Lorenzo Paoloni

Sessione II
Anno Accademico 2009/10

Indice

Capitolo 1

Introduzione

La tecnologia VoIP è una tecnologia in continua crescita, specialmente nel campo della telefonia mobile. Infatti, l'introduzione di apparecchi di telefonia, quali smartphone, sempre più complessi e dotati di più interfacce di rete eterogenee, ha reso possibile l'affacciarsi di ciò che un tempo era semplicemente considerato un apparecchio telefonico al vasto mondo di Internet, rendendo quindi possibile la capacità di effettuare telefonate tramite il sistema VoIP.

Per quanto il combinare un sistema di telefonia basato su indirizzo IP, quale VoIP, con dispositivi multihomed risulti una soluzione vantaggiosa, è bene ricordare che tutto ciò è comunque soggetto a varie problematiche e limitazioni, come la capacità di utilizzo di più di un indirizzo IP (dispositivo multihomed) nella stessa sessione in un contesto IP-based (sistema VoIP) e la possibile difficoltà a mantenere i giusti requisiti QoS durante la transazione da una rete all'altra (handover).

Si necessita quindi di un architettura in grado di superare questi ostacoli e fornire un'adeguata QoE (Quality of Experience) all'utente mobile. Tale architettura è denominata Always Best Packet Switching; un modello di infrastruttura distribuita. ABPS definisce un protocollo che estende SIP/RTP permettendo l'identificazione e l'autenticazione mediante un identificatore, univoco per ogni utente e indipendente dalla rete di provenienza. In particolare, oggetto di questa tesi, è stata l'implementazione e lo sviluppo di un sistema ABPS su un dispositivo mobile dotato di sistema operativo Symbian. L'intera fase di progettazione è stata eseguita sulla base di alcune modifiche apportate a due fondamentali librerie, in grado di essere facilmente integrate su un sistema Symbian OS. PJSIP per l'implementazione dei protocollo SIP ed RTP/SRTP, LIBZRTP per lo sviluppo di un sistema key-agreement a supporto di RTP/SRTP. L'intera progettazione, compresa la fase di testing, è stata svolta tramite un particolare ambiente di sviluppo free, messo a disposizione dalla Symbian Foundation, Carbide.c++. Un SDK integrato per carbide.c++ (nello specifico la

versione S60) ed un simulatore di dispositivi mobile ad esso connesso. Perchè il dispositivo mobile potesse essere testato in un contesto ABPS si è reso necessario l'utilizzo di una macchina server in grado di comunicare con il client, sulla quale è stato montato un opportuno proxy server in grado di gestire un traffico VoIP (sia a livello SIP sia a livello stream media, ovvero RTP/SRTP). Nello specifico si tratta del software siproxd (versione 0.7.1), opportunamente modificato, per adempiere alle specifiche richieste dal sistema ABPS.

Nei primi due capitoli verranno presentati a grandi linee i protocolli e le attuali tecnologie che costituiscono lo scheletro di un sistema VoIP, le problematiche dovute al multihomed e la soluzione adottata: ABPS. Nei restanti capitoli si affronterà il discorso sicurezza: dai basamenti fino alle tecniche utilizzate per garantire sicurezza durante una sessione multimediale VoIP. Infine, dopo una prima descrizione delle librerie PJLIB e LIBZRTP, si passerà alla fase di implementazioni e progettazione di queste su un sistema operativo Symbian.

Capitolo 2

Protocolli e Tecnologie a supporto dei sistemi VoIP

2.1 Introduzione al sistema VoIP

La nascita del VoIP (1) si fa risalire al 1995, quando alcuni appassionati informatici israeliani sperimentarono la prima comunicazione voce tra due computer. Nello stesso anno la tecnologia fu inserita in un software noto come "Internet Phone Software": per parlare da PC a PC era sufficiente un modem, una scheda audio, autoparlanti e microfono. Nel 1998, in particolare, furono introdotti i primi gateway di interconessione per abilitare le chiamate da PC alle linee telefoniche tradizionali.

Una continua crescita nel mercato della telefonia mobile (si pensi solo che nell'anno compreso tra il 2004 ed il 2005 la vendita di smart-phone è raddoppiata) ha portato ad un maggior sviluppo dei protocolli e delle tecnologie VoIP; infatti, al giorno d'oggi, tale sistema si è largamente ampliato dando la possibilità di estendere il servizio anche ad apparecchi mobile, quali appunto smart-phone, dotati di uno o più dispositivi che permettano una connessione ad internet.

Con il termine VoIP (Voice over IP) si intende un insieme di tecnologie e protocolli che rendono possibile effettuare delle conversazioni telefoniche, usufruendo di una connessione ad Internet o un'altra rete dedicata in grado di sfruttare il protocollo IP, come, ad esempio, una rete LAN interna ad un edificio o ad un gruppo di edifici. I pacchetti dati, contenenti informazioni vocali e codificati in forma digitale, vengono instradati sulla rete nel momento del bisogno, ovvero, quando uno degli utenti collegati sta parlando.

Tra vantaggi da annoverare rispetto alla telefonia tradizionale che il protocollo VoIP porta, è

bene ricordare:

- Un minor costo delle chiamate (specialmente su lunghe distanze): lo scenario che appoggia pienamente questo vantaggio, è quello che si realizza nella comunicazione tra utenti connessi tramite dispositivi (es. PC, smart-phone) ad una rete internet, in cui la comunicazione non inizia e/o non termina su normali utenze telefoniche. Pertanto, nulla è dovuto ai gestori telefonici per realizzare la comunicazione tra le parti e, tolto il costo già sostenuto da entrambi gli utenti per accedere ad Internet, le chiamate risultano assolutamente gratuite.
- Un minor costo delle infrastutture: la possibilità di far leva su risorse di rete preesistenti consente una notevole riduzione dei costi sia a livello privato che aziendale, nello specifico per quanto riguarda le spese di comunicazioni interaziendali e tra sedi diverse.
- L'aggiunta di nuove funzionalità avanzate ed estensione del protocollo senza sostituzione dell'hardware preinstallato: negli anni il sistema VoIP si è evoluto notevolmente, rendendo possibile l'aggiunta di nuove funzionalità altrimenti non disponibili nella normale rete telefonica, quale la chiamata in attesa, chiamata a tre, deviazione di chiamata ed audioconferenza. Un sistema versatile che permette un futuro arricchimento con nuove funzionalità, tramite l'installazione di nuovi moduli software o upgrade di quelli già esistenti.

La tecnologia Voice over IP richiede l'uso di due tipologie di protocolli di comunicazione in parallelo: una per il trasporto dei dati e la descrizione delle sessioni multimediali SDP (Session Description Protocol) ed una per la segnalazione della conversazione. Per il trasporto dei dati il protocollo maggioremente utilizzato è RTP (Real-time Transport Protocol) mentre la gestione delle chiamate, attualmente, verte su due diverse proposte, supportate rispettivamente da due grandi enti internazionali di standardizzazione: l'ITU (Internet Telecommunications Union) a supporto del protocollo H.323 e l'IETF (Internet Engineering Task Force) a supporto del protocollo SIP. Nonostante la diffusione dello standard H.323, il mercato dei dispositivi per l’utenza finale, del software e dei gateway per la connessione alla rete PSTN (Public Switched Telephone Network) si è ormai orientato

verso soluzioni basate su SIP. Quasi tutti i prodotti sul mercato sono compatibili con entrambi gli standard e numerosi consorzi ed enti di standardizzazione, tra cui il 3GPP per UMTS, hanno incluso SIP nelle loro specifiche.

2.2 Il protocollo IP

VoIP, come dice il nome stesso, è una tecnologia la cui base è il protocollo IP, costituente le fondamenta di Internet. L'Internet Protocol (IP) (2) è un protocollo di rete a commutazione di pacchetto che prevede la trasmissione di blocchi (chiamati datagrammi o pacchetti) includendo anche, in caso di datagrammi di grosse dimensioni, una possibile frammentazione e riassemblaggio di quest'ultimi, con una conseguente trasmissione di "small packet" sulla rete provenienti da fonti e destinazioni rappresentate da host. Il tutto però senza però fornire un supporto per quanto riguarda l'affidabilità, il flusso, o la sequenza dei datagrammi, compito lasciato gestire da altri protocolli di livello superiore. L'indentificazione degli host avviene tramite l'assegnazione di un indirizzo a lunghezza fissa (indirizzo IP), nello specifico, è bene puntualizzare che l'indirizzo IP non è assegnato all'host fisico (computer, server o altri tipi di terminale) bensì ad ogni interfaccia di rete, disponibile sull'host, in grado di effettuare una comunicazione esterna verso Internet. Tale assegnazione avviene tramite l'uso di diversi componenti e protocolli di seguito spiegati brevemente:

- DNS (Domain Name System): realizzato tramite un database distribuito, rappresenta una delle caratteristiche più "visibili" di Internet, associando un indirizzo IP ad un nome simbolico e creando vantaggi per la memorizzazione di una fonte, sia a livello macchina (si pensi ad un sito Word Wild Web) sia a livello umano.
- Subnet Mask: metodo utilizzato per definire il range di appartenenza di un host all'interno di una rete IP, al fine di ridurre il traffico di rete e facilitare la ricerca di un determinato indirizzo ip della stessa.

- Gateway o Router: questi dispositivi possiedono più interfacce e collegano tra loro sottoreti diverse, inoltrando pacchetti IP da una all'altra. Per decidere su quale interfaccia inviare un pacchetto ricevuto, cercano l'indirizzo destinazione del pacchetto in una tabella di routing, che nei casi non banali viene gestita dinamicamente tramite uno o più protocolli di routing.

Attualmente esistono due versioni distinte del protocollo IP: IPv4 ed IPv6. La prima rappresenta la versione corrente in uso, denominata cosi per distinguerla dalla nuova versione IPv6 nata principalmente per soddisfare le esigenze dovute al crescente numero di terminali connessi ad Internet.

2.3 Il protocollo SIP

Alla base di una comunicazione VoIP risiede l'instaurazione e la gestione di una sessione, dove per sessione si intende uno scambio di dati tra due o più partecipanti. L'attuazione di questa procedura è complicata da parte dei partecipanti stessi: gli utenti possono muoversi tra gli endpoint indirizzabili da più nomi diversi e che comunicano tramite media diversi – a volte contemporaneamente.

Il protocollo Session Initiation Protocol (SIP) (3) è un protocollo di livello "Applicazione" che sfrutta i livelli sottostanti, in particolare UDP (recenti revisioni dello standard permettono anche il suo utilizzo tramite TCP e TLS), il quale si occupa di gestire in maniera generale una sessione tra due o più entità, consentendo ad un endpoint UA (User Agent) di concordare una caretterizzazione (creazione, modifica e rilascio) di una sessione con un altro endpoint UA. Per la localizzazione di potenziali partecipanti alla sessione e per altre funzioni, SIP consente la creazione di un infrastruttura di host di rete a cui gli UA possono inviare immatricolazioni, bandi di sessione e altre richieste. Le principali componenti logiche appartenenti a queste infrastrutture sono:

- SIP Registrar Server: Componente rappresentante un ancoraggio alla rete per un UA

il quale richiede, tramite un messaggio di REGISTER, la sua registrazione al servizio SIP (fondamentale per poter essere rintracciato da altri utenti SIP) basata sull'associazione tra indirizzo IP del richiedente ed indirizzo permanente email-like SIP. Il registrar si occuperà, una volta accettata la richiesta, di salvare questa associazione all'interno del suo user location database. Tale database verrà aggiornato ad ogni richiesta di registrazione di un nuovo User Agent.

- SIP Redirect Server: A richiesta dell'UA chiamante fornisce a quest'ultimo l'indirizzo dell' UA destinatario della chiamata. Ricevuto l'indirizzo il chiamante può effettuare un collegamento autonomo.

- SIP Proxy Server: Con funzionalità di intermediario, può rispondere direttamente alle richieste oppure inoltrarle ad un client, ad un server o ad un ulteriore proxy. Un proxy server analizza i parametri di instradamento dei messaggi e "nasconde" la reale posizione del destinatario del messaggio - essendo quest'ultimo indirizzabile con un nome convenzionale del dominio di appartenenza (SIP address email-like).

In figura (2.1) sono raffigurate le fasi fondamentali per l'instaurazione di un canale SIP, partendo da una registrazione, fino ad arrivare alla chiamata da parte di un chiamante ad un chiamato, tutto mediante il protocollo SIP.

1. Come prima operazione (fase 1 e 2) l'UA, per poter essere contattato, dovrà registrarsi al servizio SIP disponibile, effettuando una REGISTER al SIP Registrar Server e ricevendo una conferma (msg 200 OK) da quest'ultimo. Una volta avvenuta la registrazione, il contatto risulterà disponibile a ricevere ed effettuare chiamate.
2. Il chiamante inoltra una chiamata tramite l'invio di un messaggio di INVITE (fase 3), il SIP Proxy Server, che funge da intermediario momentaneo, verifica la presenza dell'indirizzo di destinazione (fase 4), se il controllo è andato a buon fine (l'utente richiesto esiste ed é regolarmente registrato) ridirige l'INVITE al destinatario, il quale decide se accettare la chiamata o no (fase 5 e 6), in caso positivo inizia una fase di

scambio (il Proxy server funge ancora da intermediario) di ACK per confermare il collegamento diretto (fase 6, 7 e 8).

3. Entrambi gli User Agent (sia chiamato sia chiamante) entrano in una comunicazione diretta, il server Proxy non svolge più da intermediario e SIP lascia la gestione della comunicazione, con il trasporto di dati real-time audio e/o video, al protocollo RTP che verrà descritto in seguito.

4. L'ultima fase è la fase di fine chiamata (fase 11 e 12), il controllo ripassa al protocollo SIP, uno dei partecipanti decide di terminare la comunicazione mandando un messaggio SIP di BYE e il destinatario risponde con un 200 OK di avvenuta ricezione del messaggio.

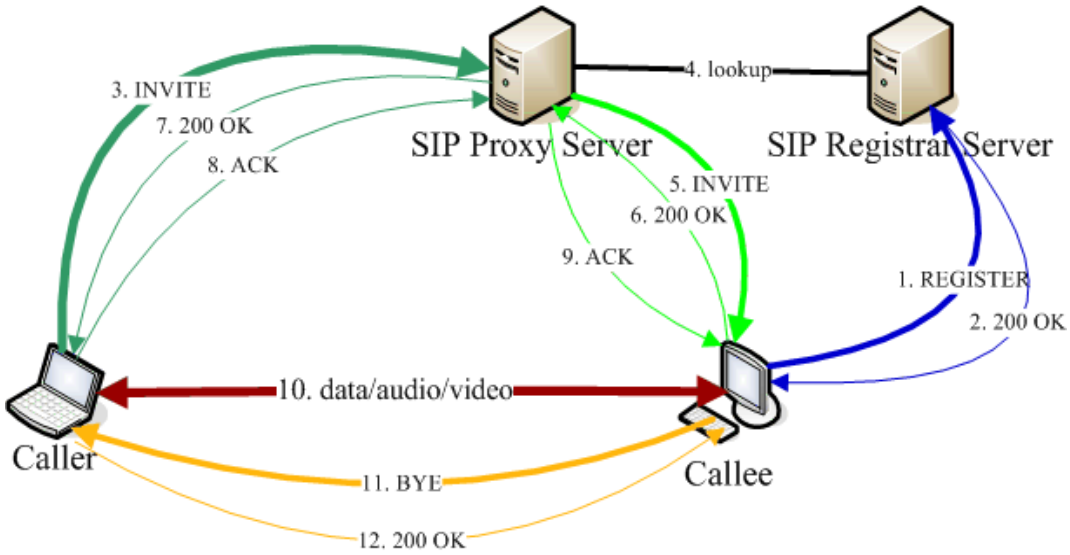

Figura 2.1 – Fase di registrazione di un client ad un provider SIP e successiva instaurazione di una sessione.

2.4 Il protocollo RTP ed SRTP

Come descritto prima nel punto 3 della creazione di una sessione, finita la fase di instaurazione di essa, SIP lascia gestire la comunicazione real-time diretta ad un altro tipo di protocollo, il Real-time Transport Protocol (RTP) (4). RTP è un protocollo di comunicazione che fornisce funzioni di trasporto di rete end-to-end adatte per le applicazioni di trasmissione di dati in tempo reale, come ad esempio: audio, video o dati di simulazione, oltre a servizi di rete multicast, con cui si indica la distribuzione simultanea di informazione verso un gruppo di destinatari, ed unicast, distribuzione di informazione ad un unico destinatario. RTP non affronta la fase di prenotazione delle risorse (fase assegnata a SIP) né garantisce qualità per i servizi real-time, quali la garanzia di consegna in tempo utile, ma si basa sull'affidabilità dei servizi di livello inferiore. Il protocollo può avvalersi di un servizio aggiuntivo, nello specifico il protocollo RTCP (Real-time Transport Control Protocol) (5), per il monitoraggio della trasmissione dei dati in modo scalabile per reti multicast di grandi dimensioni e per fornire un controllo minimo e funzionalità di identificazione. Solitamente le applicazioni pongono l'RTP sopra l'UDP per le operazioni di multiplexing e checksum, anche se può essere usato con altri protocolli di rete e trasporto sottostanti.

La figura 2.2 rappresenta un header RTP aggiunto ai dati del flusso multimediale ed inviato come payload del livello di trasporto.

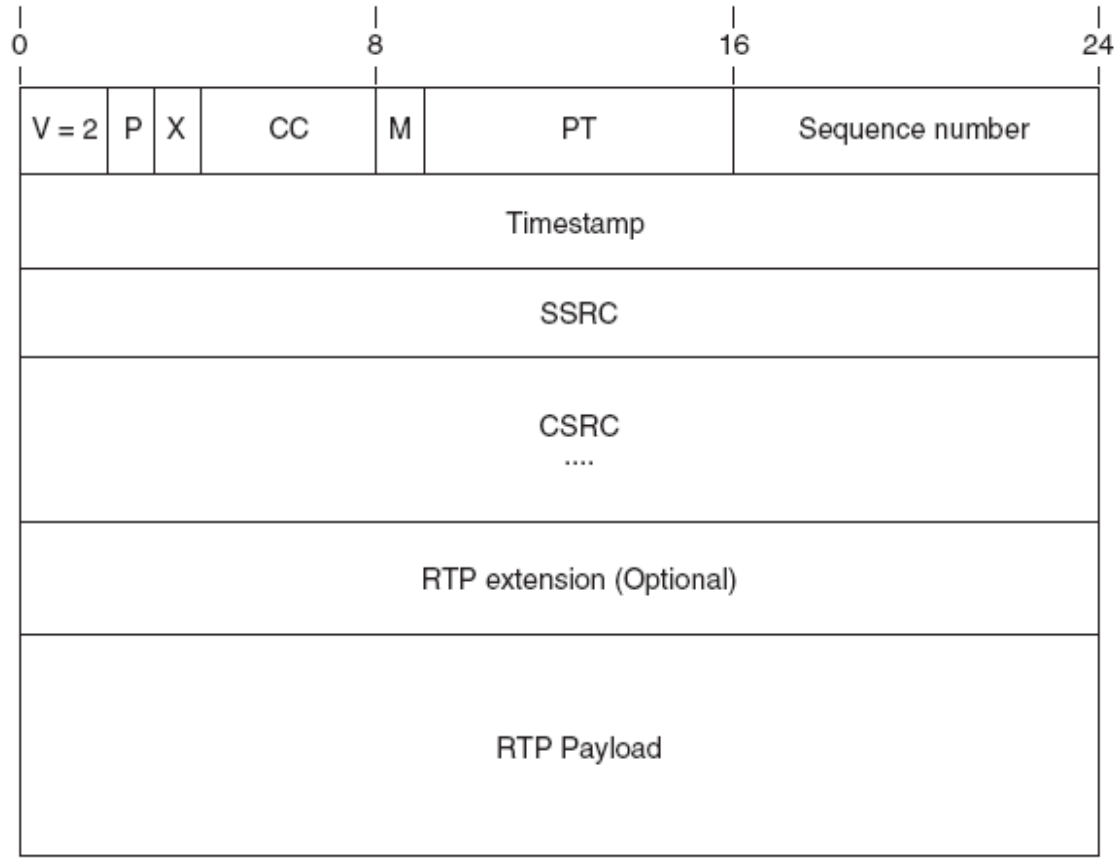

Fig 2.2 Header RTP

La sicurezza è parte fondametale e sicuramente da non trascurare in una sessione multimediale. Uno scenario costituito da due o più partecipanti ad una comunicazione multimediale in cui si presenta una terza parte, all'insaputa dei partecipanti, in grado di intercettare i flussi multimediali, registrarli o addirittura ascoltarli direttamente nel momento in cui avviene la discussione, non si deve assolutamente presentare. RTP, per quanto effettui un ottima pacchettizzazione di flussi audio e/o video, non garantisce nessun tipo di integrità e sicurezza dei dati. Nasce quindi l'esigenza di introdurre un sistema che

garantisca piena sicurezza tramite l'utilizzo di tecniche quale la crittografia dei dati multimediali e che possa soddisfare una serie di requisiti fondamentali, di seguito elencati, per una comunicazione real-time:

1. Non devono essere presenti (o presenti in minima parte) aumenti nella dimensione dei messaggi inviati. E' necessario utilizzare i campi già presenti nel payload dei pacchetti RTP.
2. Gli header dei pacchetti RTP non devono essere soggetti a nessun tipo di compressione.
3. La decrittazione dei payload deve essere sempre possibile, anche quando i datagrammi arrivano non ordinati o vengono persi.
4. L'hardware necessario alla crittazione dei dati deve essere minimo. Gran parte delle applicazioni VoIP funzionano su sistemi mobile a basso consumo.

Il protocollo Secure Real-time Transport Protocol (SRTP) (6) si adatta bene a tutti i requisiti sopra elencati. Progettato per adattarsi specificatamente alle applicazioni real-time, offre confidenzialità, integrità, autenticazione e protezione replay per il traffico RTP ed RTCP. La sicurezza di SRTP si basa sull'uso di un cifrario a flusso additivo per la crittografia ed una funzione hash key-based per l'autenticazione dei messaggi, un indice implicito per il sequenziamento / sincronizzazione basato sul sequence number di RTP. In particolare l'algoritmo di default utilizzato per la crittografia e decrittografia del flusso di dati è l'AES (Advanced Encryption Standard). Utilizzato in due possibili diverse modalità:

1) Segmented Integer Counter Code. Un contatore classico (solitamente le funzioni stesse di SRTP fungono da contatore) permette l'accesso causale ai pacchetti RTP da spedire. In questa modalità AES opera come di default: utilizzando una chiave da 128 bit ed una salt key da 112 bit. Il difetto di questa modalità sta nella possibile perdita durante la fase di crittografia di alcuni pacchetti.
2) f8-mode. Rappresenta una variante dell'output feedback mode. Tutti i valori utilizzati da AES sono gli stessi utilizzati nella modalità counter.

SRTP oltre ad utilizzare AES per la crittografia dei dati, è in grado anche di applicare una sorta di algoritmo nullo che rappresenta niente altro che un annullamento della crittografia

sui pacchetti RTP.

Anche se efficace, AES su SRTP non garantisce piena integrità ed autenticità dei messaggi. Per far ciò si utilizza un algoritmo HMAC-SHA1 calcolato sul payload e l'intestazione (compreso il numero di sequenza) del pacchetto RTP da spedire. L'output di 160 bit verra troncato in un output di 80 o 32 bit e successivamente allegato come tag al pacchetto RTP uscente.

Il problema che presenta SRTP è l'impossibilità di adoperare un proprio algoritmo di key-agreement (in quanto completamente spovvisto) necessario ad effettuare uno scambio di chiavi-segrete (generalmente chiamate master-key) costituenti le chiavi utilizzate per la crittografia e l'autenticazione. Ed è per questo che si rende necessario l'utilizzo e l'integrazione di un protocollo aggiuntivo, ad esempio: SDES, MIKEY, DTLS-SRTP e ZRTP, il cui compito è la negoziazione dei parametri di sicurezza in un contesto crittografico ed il fornire i terminali di flusso di una coppia di segreti condivisi ("master-key" e "master-salt").

2.5 Il protocollo di trasporto UDP

Sebbene non facente parte strettamente dei protocolli VoIP, è bene ricordare che SIP ed RTP, fin qui discussi, lavorano al di sopra di un protocollo di trasporto a pacchetto: l'User Datagramm Protocol (7). L'UDP , usato solitamente in combinazione con il protocollo IP, è un protocollo di tipo connectionless, con il quale si intende un protocollo che permette uno scambio di dati tra una sorgente ed un destinatario senza la necessità di un'operazione preliminare per la creazione di un circuito, fisico o virtuale, su cui instradare l'intero flusso di dati in modo predeterminato ed ordinato nel tempo (sequenziale). Più semplicemente i dati vengono suddivisi in pacchetti inviati in maniera indipendente l'uno dall'altro, senza interazioni di ritorno tra sorgente e destinatario (per esempio per verificare se il destinatario è raggiungibile) e senza controllo sulla corretta sequenza temporale di inoltro. UDP non gestisce il riordinamento dei pacchetti né la ritrasmissione di quelli persi, fattore che lo

differenzia da un altra tipologia di protocolli di trasmissione denominati connection-oriented, nella quale rientra TCP (Transmission Control Protocol), ed è perciò generalmente considerato di minore affidabilità.

Il motivo per cui viene utilizzato UDP come livello di trasporto sottostante a SIP ed RTP, è dato dal fatto che tale protocollo è ideale per le applicazioni real-time che, molto spesso, richiedono un ritmo minimo di spedizione, non vogliono ritardare eccessivamente la trasmissione dei pacchetti e possono tollerare qualche perdita di dati. UDP è un protocollo stateless (ovvero non tiene nota dello stato delle connessioni) e fornisce soltanto i servizi basilari del livello di trasporto: multiplazione delle connessioni, tramite l'utilizzo delle porte e verifica degli errori mediante checksum, inserita nel campo di intestazione del pacchetto (figura 2.3).

+	**Bit 0-15**	**16-31**
0	Source Port (Optional)	Destination Port
32	Length	Checksum (Optional)
64	Data	

Fig. 2.3 Esempio pacchetto UDP

2.6 Wi-fi

L’abbattimento dei costi di produzione e di commercializzazione, la comodità e facilità d’uso e d’installazione, la possibilità di usufruire di connettività mobile hanno portato in

breve tempo ad un'inevitabile e stretta associazione tra sistema VoIP e reti Wi-fi (Wireless Fidelity) (8). Sostanzialmente Wi-Fi è il termine con il quale si indicano dispositivi in grado di collegarsi a reti locali senza fili (WLAN) basate sulle specifiche IEEE 802.11. Le reti Wi-Fi sono infrastrutture relativamente economiche e di veloce attivazione in grado di realizzare sistemi flessibili per la trasmissione di dati usando frequenze radio, estendendo o collegando reti esistenti, ovvero, creandone di nuove.

L'architettura internet è del tutto simile ai tradizionali ISP, che forniscono un punto di accesso agli utenti che si collegano da remoto.

La fonte di connettività a banda larga può essere via cavo (ADSL o HDSL) oppure via satellite. Oggi esistono connessioni a internet satellitari bidirezionali, che consentono alte velocità di trasferimento dei dati sia in download che in upload. La trasmissione satellitare ha, tuttavia, tempi di latenza elevati; il tempo di attesa prima che inizi l'invio dei pacchetti è dell'ordine di 1-2 secondi, un tempo molto grande se confrontato ai pochi centesimi di secondo necessari ad una connessione DSL.

La copertura Wi-Fi, permessa da dispositivi quali antenne, è generalmente di due tipi: omnidirezionali e direttive.

Le antenne omnidirezionali vengono utilizzate di norma per distribuire la connettività all'interno di uffici o comunque in zone private e relativamente piccole, oppure, con raggi d'azione più grandi, si possono coprire aree pubbliche (come aeroporti, centri commerciali ecc...). Con le antenne direttive è invece possibile coprire grandi distanze, definibili in termini di chilometri, e sono utili proprio per portare la banda larga nei territori scoperti dalla rete cablata. In questo caso, è possibile aggregare più reti in un'unica grande rete, portando la banda in zone altrimenti scollegate.

Al giorno d'oggi molti dispositivi mobile offrono la possibilità di connessione tra numerose interfacce disponibili: GPRS, UMTS ed altro ancora. Ma sicuramente, dati i vantaggi, specialmente legati ai costi del servizio, la WiFi rimane in assoluto la tecnologia più utilizzata per i sistemi VoiP su dispositivi mobile. Un primato che probabilmente in futuro verrà lasciato ad un nuovo sistema, una sorta di evoluzione di tale tecnologia: la WiMax (Worldwide Interoperability for Microwave Access). Una tecnologia basata sulle specifiche IEEE 802.16 (non ancora molto diffuse) che consente l'accesso a reti di telecomunicazioni a

banda larga e senza fili, offrendo una maggior e decisamente più vasta copertura sul territorio, oltre ad apportare migliorie legate alla velocità di trasmissione dei dati.

2.7 Il concetto di Multihoming

Con il concetto di multihoming si intende un dispositivo che fa uso di più indirizzi IP associati a varie reti collegate. All'interno di questo scenario, l'host multihomed è fisicamente legato ad una varietà di connessione dati associate a differenti interfacce di reti.

Al verificarsi di una problematica di connessione riguardante un interfaccia (guasto nella rete, fallimento nell'instaurazione del collegamento, scollegamento causato da un basso segnale di recezione), un dispositivo multihomed è in grado di passare ad un altra interfaccia di rete disponibile e funzionante. Inoltre, un dispositivo multihomed che opera tramite un protocollo VoIP SIP su rete wireless, è in grado di usufruire di uno specifico adattamento e bilanciamento del traffico (pacchetto per pacchetto) in relazione alla fluttuazione della disponibilità di banda e della latenza nella rete.

Per quanto i vantaggi di un dispositivo multihomed su un sistema VoIP ci siano, è bene non trascurare le problematiche e limitazioni che ne seguono. Come già detto in uno scenario IP-based il verificarsi di più indirizzi IP presenti su un unica sessione ed il mantenimento dei parametri QoS (Quality Of Service) (9) durante una possibile transazione da una rete ad un altra creano delle limitazioni che devono essere affrontante con un giusto sistema. Nel prossimo capitolo verrà descritta una possibile soluzione, il sistema ABPS che tramite la definizione di un protocollo, estensione dei già preesistenti SIP/RTP, permette l'identificazione e l'autenticazione mediante un identificatore univoco indipendente dall'interfaccia di rete su cui un dispositivo opera.

2.8 Scenario Multihoming

Precedentemente si è spiegato il concetto di multihoming, come esso può essere esteso ad apparecchi mobile: personal-computer o cellulari di nuova generazione, quali gli smart-phone, dotati di più di un interfaccia di rete wireless ed i vantaggi e svantaggi che esso porta.

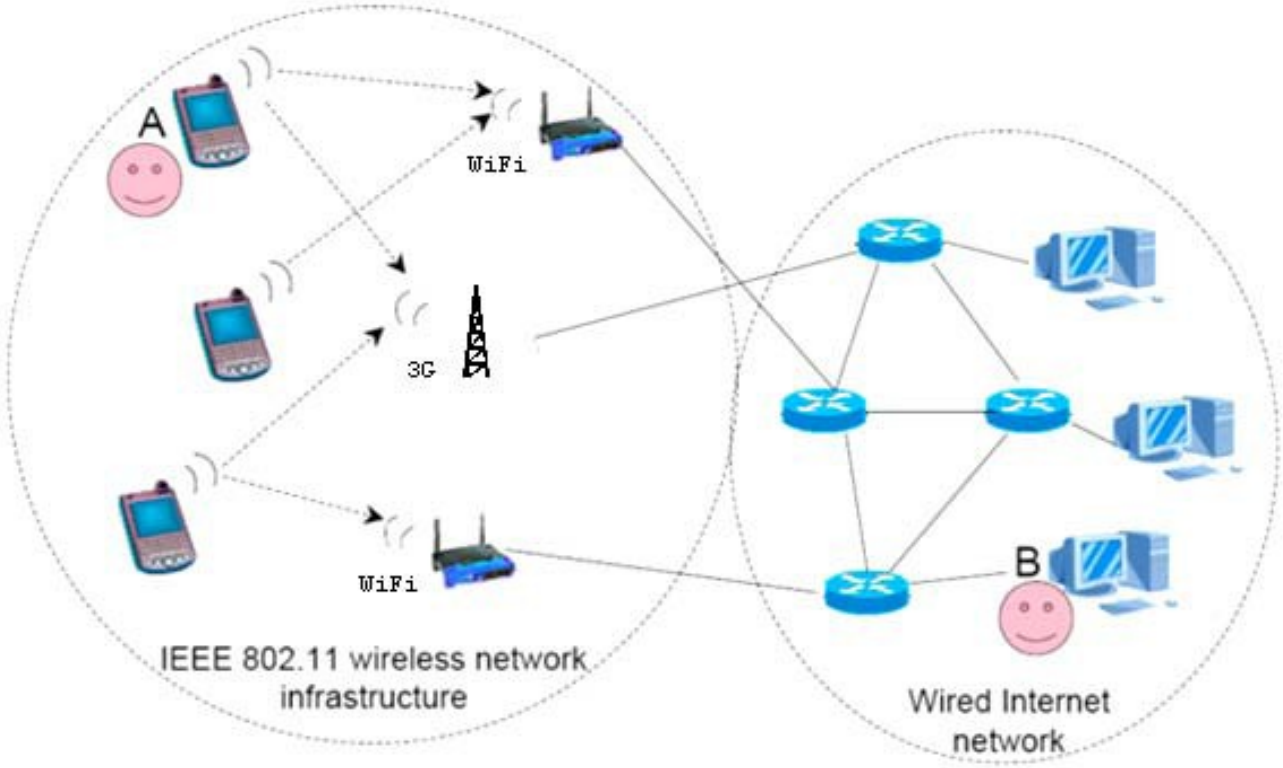

Fig. 2.4 Scenatio Multihoming Handover

In figura 2.4 è rappresentato un tipico scenario multihoming handover, in particolare si hanno due terminali A e B, rispettivamente uno smart-phone dotato di due interfacce di rete (WiFi standard 802.11 b/g/n e 3G) ed un terminale dotato di un unica interfaccia di rete (WiFi sempre standard 802.11 b/g/n), collegati alla rete per una comunicazione VoIP. Il cambio di access point e di interfaccia di rete è cosa comune, si pensi ad un ambiente urbato in cui un utente effettua spostamenti quotidiani rendendo il cambio di copertura di rete

obbligatorio. Si rende quindi necessario l'utilizzo del concetto di multihoming, dove, per esempio, il problema della mancanza di un segnale specifico ad un'interfaccia di rete, dovuta a varie problematiche, viene risolta tramite l'estensione dell'interfaccia WiFi con una copertura GPRS/UMTS permettondo al dispositivo una continua connettività, il tutto gestito da un metodo che garantisce nel miglior modo lo switch di interfaccia di rete: ABC (Always Best Connected). ABC è uno speciale framework che gestisce nel miglior modo la scelta del miglior NIC (Network Interface Controller) disponibile sul momento da selezionare in caso di degradamento dell'attuale in uso. L'architettura Always Best Packet Switching si basa su questo modello, ponendosi come scopo quello di offrire all'utente mobile il massimo della qualità di comunicazione sfruttando al meglio le capacità aggregate di tutte le interfacce di rete disponibili.

Capitolo 3

La sicurezza

3.1 Il concetto di Sicurezza

La sicurezza è quella branca dell'informatica che si occupa della salvaguardia dei sistemi informatici da potenziali rischi e/o violazione dei dati. I principali aspetti di protezione del dato sono: confidenzialità, integrità e disponibilità.

Di seguito, nella prima parte, verranno presentati i principali algoritmi di crittografia necessari a garantire confidenzialità ed integrità, meccanismi di autenticazione ed alcuni metodi per eseguire Key-agreement (scambio di chiavi). Successivamente verrà presentato un particolare protocollo di sicurezza per eseguire key-agreement, ZRTP, in quanto protocollo utilizzato in appoggio a SRTP, per adempiere ad un sistema di sicurezza in ABPS (scambio di chiavi e crittografia dei pacchetti).

3.2 MD5, SHA-1, HMAC-MD5, HMAC-SHA-1

Con il concetto di crittografia si intende un sistema di metodi in grado di "offuscare" un messaggio in maniera da non poter essere letto da persone non autorizzate. Tra i principali algoritmi di crittografia da ricordare, utilizzati anche durante lo sviluppo della sicurezza su ABPS troviamo: MD5, SHA-1 ed HMAC.

L'MD5 (Message Digest algorithm 5) (10) indica un algoritmo crittografico di hashing, il quale prevede un input costituito da una stringa di dimensione arbitraria seguito da un output avente un altra stringa di lunghezza 128 bit (chiamato MD5 checksum o MD5 hash) che può essere facilmente utilizzata per calcolare la firma digitale, inoltre, la codifica dell'output restituito avviene molto velocemente ed è tale da rendere improbabile il presentarsi di output simili. Nonostante siano presenti algoritmi di hasing più veloci ed efficienti (quali WHIRLPOOL SHA-1 o RIPEMD-160) MD5 rimane un algortimo ancora ampliamente usato, infatti gran parte dei controlli di integrità su file vengono eseguiti tramite MD5. Le operazioni di base su cui lavora MD5 sono:

1) Fase di Padding: al messaggio in input (di lunghezza arbitraria) da codificare vengono aggiunti un insieme di bit fino a raggiungere la dimensione pari a 448 (mod 512). In particolare il primo bit aggiunto è 1 ed i restanti tutti 0.
2) Aggiunta della lunghezza: viene aggiunta una rappresentazione a 64 bit della lunghezza del messaggio prima che avvenga il padding. Nel caso in cui la lunghezza superi i 2^{64} , vengono utilizzati solamente i 64 bits meno significativi del messaggio. Questi 64 bits rappresentano due parole da 32 bits ciascuna che vengono poi appese al seguito del messaggio in input, dando la precedenza alla parola meno significativa. Si ottiene quindi un messaggio di lunghezza multipla di 512 bits (16 parole da 32 bits ciascuna).
3) Nella terza fase avviene l'inizializzazione del buffer costituito da 4 words aventi ciascuna dei valori esadecimali di inizializzazione.

4) Nell'ultima fase vengono eseguite quattro di funzioni logiche sull'insieme delle sedici parole ottenute nelle prime due fasi iniziali e sui quattro registri del buffer. In quanto rappresenti una fase lunga da spiegare, verranno semplicemente elencate le 4 funzioni tralasciando il procedimento logico-matematico:

   F(X, Y, Z) = (X and Y) or (not X and Z).

   G(X, Y, Z) = (X and Z) or (Y and not Z).

   H(X, Y, Z) = X xor Y xor Z.

   I(X, Y, Z) = Y xor (X or not Z).

   Le funzioni non operano sulle parole intere ma direttamente bit per bit. Ogni singola parola viene passata su ogni differente funzione.

   Come output si ottengono 4 parole da 32 bits che unite, dando sempre la precedenza alla parola meno significativa, compongono l'hash di 128 bit.

Più efficiente ma di minor impiego risulta l'SHA-1 (Secure Hash Algorithm) (11), il più diffuso di una famiglia composta da cinque possibili varianti dell'algoritmo: SHA-1, SHA-224, SHA-256, SHA-384 e SHA-512. E' basato su un sistema simile all'MD5 se non per il fatto che produce un digest di 160 bit ricavato da un messagio in input di massimo 2^{64}-1 bit. Il principio di funzionamente dell'SHA-1 è simile a quello dell'MD5.

L'HMAC (Keyed-hash Message Authentication Code) (12) si differenzia dagli altri algoritmi crittografici per due fattori:

1) Non possiede una prorpia funzione per il calcolo hash. HMAC viene solitamente utilizzato in combinazione con MD5 o SHA-1 (HMAC-MD5 e HMAC-SHA-1), quindi, il procedimento di calcolo dell'hash è strettamente legato alla tipologia di funzione generatrice in uso.
2) Fa uso di una chiave segreta in supporto alla funzione hash utilizzata.

Si consideri H come l'algoritmo in utilizzo (MD5 o SHA-1), K una chiave segreta, m il messaggio in input e K' un output detto chiave di HMAC:

1) La stringa in input viene suddivisa in j bits. Se:

- Se |K| = j allora K'=K
- Se |K| < j, allora eseguo la procedura di padding (come in MD5), ovvero:

  K' = K + padding di zeri.
- Se |K| > j, allora applico l'algoritmo H scelto, ovvero: K' = H(K).

2) si definiscono due stringhe di bit che serviranno per eseguire uno XOR con la chiave privata: ipad=00110110 ed opad=01011010. Si esegue il passo finale per il calcolo dell'output ottenuto dalla formula:

   HMAC (K, m)=H ((K xor opad) || H((K xor ipad) || m))

3.3 Tipologie e Meccanismi di Autenticazione SIP

L'autenticazione, seguendo la definizione, è il processo tramite il quale un computer, un software o un utente verifica la corretta, o almeno presunta, identità di un altro computer, software o utente che vuole comunicare attraverso una connessione. L'autenticazione generalmente avviene prima dell'instaurazione di un canale ma, per garantire una piena sicurezza, vi è anche una fase di autenticazione che avviene quando già il canale è instaurato e le parti sono già in fase di scambio di dati.

Applicando realmente il concetto di autenticazione ad un sistema VoIP (sistemi basati su SIP) è possibile distinguere tre meccanismi di autenticazione differenti:

- Client Authentication: con il quale si intende l'obbligo di un client ad identificarsi presso un server che gestisce un determinato servizio.
- Mutual Authentication: processo simile al primo, con la differenza che l'identificazione è in maniera reciproca; non solo dal client verso il server ma anche dal server verso il client, da qui il termine mutuale.
- Message Authentication: rappresenta la fase finale dell'autenticazione. Dopo

l'esecuzione di un un processo di autenticazione client o mutuale ogni messaggio spedito viene dotato, in base alle richieste del sistema in uso, di un identificativo che garantisce l'autenticità del mittente.

In particolare verrà preso un modello basato sul concetto client-server che garantisca l'implementazione di tutti e tre i meccanismi, in quanto rispecchia perfettamente il modello ABPS le cui specifiche di sicurezza verranno descritte nel prossimo capitolo.

3.3.1 Client Authentication e Mutual Authentication

Come già detto il concetto principale per l'autenticazione client è quindi il rappresentarsi tramite una propria ed univoca identità. Parlando in un contesto client-server puramente generale, in quanto il sistema ABPS implementato richiede questo tipo di comunicazione, non si può semplicemente applicare l'identità di un client al suo indirizzo IP, in quanto, per ovvi motivi, non risultà essere un sistema sicuro (l'IP non è univoco, è facile da replicare, identifica il client ma non l'utente, ecc....), ma bisogna adottare un sistema più sofisticato, in grado di soddisfare alcuni requisiti in ambito di sicurezza; tra i vari, uno di vitale importanza, è l'evitare il furto di identità. Esistono varie metodologie più o meno sicure:

- Password in chiaro: Rappresenta il sistema più semplice di Client Authentication. Il Client provvede ad inviare la coppia user e password al server in seguito alla richiesta di autenticazione del server stesso, una volta ricevute, il server si occupa di controllare le credenziali su un file ad accesso ristretto che presenta tutti gli user registrati. Per aumentare la sicurezza, la password associata all'utente e salvata sul server è spesso argomento di una funzione hash, tale che, una volta ricevuta la password dal client, il server applica la funzione hash a quest'ultima e controlla l'output con quello salvato nel file. In quanto a sicurezza questo sistema lascia a desiderare; è soggetto a molti possibili attacchi, come lo sniffing sulla rete eseguito da una terza parte in grado di catturare e decriptare l'hash contenente la password. Attacchi di tipo dizionario offline, che avvengono specialmente se il

server non gestisce un controllo su il numero di tentativi da parte del client per l'inserimento della password, dove l'intruso, tramite l'utilizzo di uno specifico dizionario, ricerca l'autenticazione forzata con richieste multiple di autenticazione al server.

- One-time Password: Il sistema è molto simile al precedente, con alcune piccole modifiche. La password usata per l'autenticazione è di tipo usa-e-getta, può quindi essere utilizzata una volta sola. Il client dispone di una serie di possibili password da utilizzare una sola volta ciascuna, le quali vengono calcolate con un apposito algoritmo. Questo sistema garantisce maggior sicurezza in quanto ad ogni identificazione dello stesso utente si associa una password sempre diversa, evitando gli attacchi di tipo replay.
- Challenge/Response: Come dice il nome è un sistema basato su una sfida (challenge), lanciata dal server, ed una risposta a tale sfida (response) data dal client. Sia server che client condividono un algoritmo hash pre-negoziato. Al momento della richiesta di autenticazione il server lancia una "sfida" al client, il quale, utilizzando una chiave segreta e la funzione hash pre-nogoziata, calcola la risposta e la invia al server per la conferma di autenticazione. La sfida tipicamente è un valore generato causalmente, in maniera che non si possa ripetere troppo spesso. Un esempio di calcolo del response può essere: RESPONSE=hash(CHALLENGE || SECRET). Questo metodo è stato progettato per due scopi: per evitare la trasmissione della password, annullando il problema del password sniffing, e per prevenire attacchi di tipo replay, assumendo che il challenge sia generato con forte entropia. Da notare infatti che, se non progettato accuratamente, il metodo challenge/response può risultare meno sicuro del metodo semplice, perché soggetto ad un attacco dizionario offline. Una soluzione adottata per evitare quest'ultimo rischio citato, è di usare un processo di hash a due passi, nel quale il server calcola il valore hash della password, utilizzando un salt, e memorizzandolo. Il server successivamente invia il challenge e il salt all’utente e aspetta il response, che verrà calcolato dal client sempre mediante un hash a due passi. Il protocollo HTTP, che utilizzava inizialmente

un’autenticazione con password in chiaro, supporta il meccanismo di challenge/response per le richieste http ai web server, così che non solo l’utente ma anche la richiesta al server sia autenticata. La procedura è chiamata HTTP digest authentication (spiegata nel prossimo paragrafo), perché un digest (hash) viene calcolato su user, password, challenge e una parte della richiesta HTTP. Questo tipo di autenticazione risulta interessante ai nostri scopi, in quanto già utilizzata dal protocollo SIP. Unico difetto di questo meccanismo è la necessità di un doppia richiesta risposta delle due parti, in quanto la prima richiesta del client viene rifiutata dal server, che invia il challenge al client, dopo di che il client può di nuovo inviare la richiesta, questa comprensiva di credenziali, raddoppiando così il traffico per le autenticazioni.

- Anonymous Key Exchange: questo meccanismo sfrutta un canale già sicuro, che garantisca integrità e privatezza dei dati scambiati, per inviare le credenziali di autenticazioni in chiaro.
- Zero-Knowledge Password Prof: sono metodi abilmente studiati per annullare i problemi dei meccanismi di autenticazione che richiedono chiavi precondivise, o password che possono essere intercettate da un man-in-the-middle (MITM), mediante uno scambio criptato delle chiavi (EKE). Questi meccanismi sono attualmente metodologie proprietarie e di conseguenza non hanno mai avuto ampia diffusione.
- Certificati Server (Mutual Authentication): questo meccanismo è basato sulla premessa che, se i due terminali (client e server) possono stabilire un canale sicuro basato su un’autenticazione unidirezionale (autenticazione di un solo terminale all’altro), l’autenticazione del secondo terminale nei confronti del primo può avvenire utilizzando il canale sicuro in un modo molto meno complesso. Qualsiasi meccanismo, anche la password in chiaro, può essere utilizzato per autenticare il secondo terminale al primo.
- Chiave Pubblica (Mutual Authentication): è il metodo più robusto e sicuro per l’autenticazione mutuale, che consiste nel presentare reciprocamente la chiave pubblica del client e del server allo scopo di autenticazione.

3.3.2 Message Authentication

I metodi che proteggono le informazioni dalla manomissione di una parte illegittima, sono definiti metodi per la protezione dell'integrità dei dati. Quando l'informazione viene trasportata nei messaggi in un canale di comunicazione, la protezione di integrità è tipicamente fornita dai meccanismi di autenticazione dei messaggi. Per fornire questa protezione, il mittente del messaggio deve dare la prova dell'autenticità dello stesso. Così come nella vita reale, dove tutti i documenti con valore legale vengono firmati dalle parti coinvolte per assicurarne l'autenticità, nelle comunicazioni digitali il mittente firma il messaggio con un segreto e aggiunge la firma alla fine del messaggio, in modo tale che questa firma dipenda sia dal messaggio che dal segreto, condiviso con la parte ricevente. In questo modo, se un man-in-the-middle modifica il contenuto del messaggio, senza essere a conoscenza del segreto, non potrà riprodurre la giusta firma corrispondente al contenuto. Per produrre questa firma digitale, il mittente deve utilizzare una funzione che prenda come input il contenuto del messaggio e la chiave. Poiché utilizzare un algoritmo di criptografia su un intero messaggio è solitamente dispendioso in termini computazionali e quindi di tempo, il mittente comprime i dati utilizzando un algoritmo hash (H) e ottiene un valore di digest, chiamato message authentication code (MAC). Il valore MAC viene poi aggiunto dal mittente alla fine del messaggio e inviato all'altra parte che ne verificherà la correttezza. Nel corso degli anni sono state sviluppate diverse funzioni di hash per l'autenticazione dei messaggi, ma il meccanismo sicuramente più standardizzato per produrre dei digest MAC è quello dell'HMAC.

3.3.3 HTTP Digest Authentication

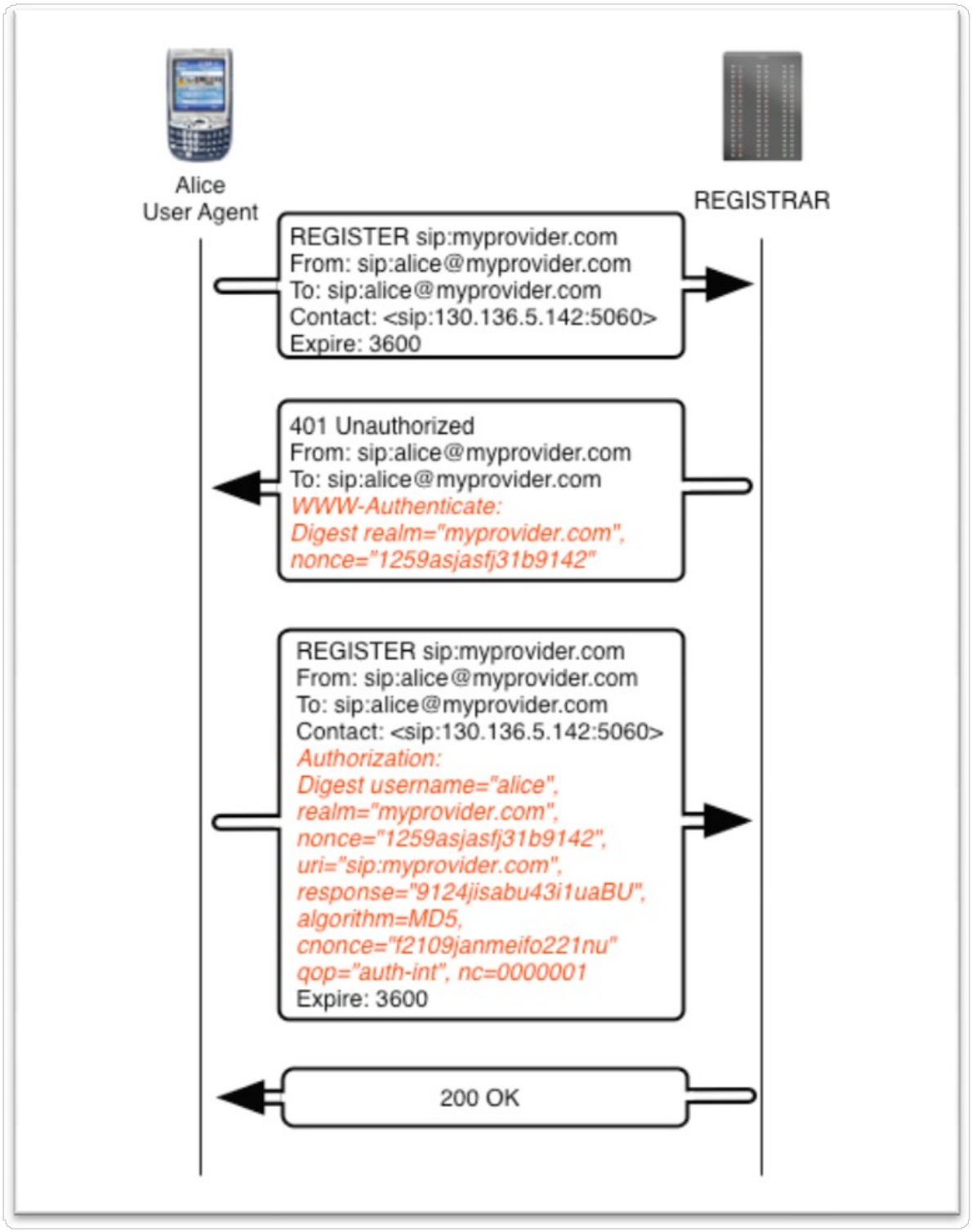

Fig. 3.1 Four-way handshake HTTP Digest

In ambito SIP, ma non solo, il modo per garantire autenticazione é quella di utilizzare un meccanismo basato sul concetto di challenge/response, L'HTTP Digest Authentication (13) (figura 3.1). Questo è un motivo per cui si è scelto di dedicare un paragrafo intero a questa tecnica di autenticazione, in quanto è stata la base su cui sono state sviluppate le idee progettuali per la sicurezza ed autenticazione del sistema ABPS.

L'HTTP Digest Authentication è basato su una verifica crittografica di una password in chiaro condivisa tra client e server. E' un protocollo challenge-response, dove il client che richiede il servizio viene sfidato a dimostrare di essere in possesso delle credenziali e che utilizza la funzione hash MD5, per permettere al client di trasmettere la chiave in canali non sicuri. Il server può verificare l'hash della password, mentre un man-in-the-middle, una volta intercettato l'hash, non potrà risalire alla password in chiaro. Il client SIP calcola un valore hash a 128-bit, utilizzando la password condivisa con il server e usando l'algoritmo MD5. Se la verifica finisse qui, un man-in-the-middle potrebbe semplicemente inviare al server l'hash intercettato per garantirsi l'accesso ma, per evitare questo tipo di attacco, ovvero un attacco di tipo replay, il server invia nel challenge un valore unico (un hash MD5) per il messaggio e non prevedibile, che sarà utilizzato dal client per generare il response.

L'autenticazione digest di una registrazione SIP è una procedura formata da 4 fasi (*four-way handshake*):

1. Richiesta iniziale: il SIP User Agent Client (UAC) invia una richiesta (e.g. REGISTER), per la quale è necessaria l'autenticazione, al server (e.g. il SIP Registrar).
2. Challenge: il server sfida il client rispondendo con un errore 401 (*Unauthorized*), o 407 nel caso si tratti di un server proxy, contenente un header *WWWAuthenticate* che include i paramentri:
    a. Realm: è il real name, contiene il nome dell'host o del dominio del server che richiede l'autenticazione ed è utilizzato per far capire al client quale coppia username/password debba usare.
    b. Quality of protection (qop): può essere "auth" o "auth-int", o può non esserci del tutto, e influenza il modo in cui il response deve essere calcolato, come vedremo successivamente.

c. Nonce: rappresenta il challenge generato dal server ed è utilizzato dal client per il calcolo del response. Sarà denotato come *server nonce*, per distinguerlo dal *client nonce*. Il server nonce consiste in un timestamp concatenato con un hash, calcolato quest'ultimo sul timestamp e su una chiave segreta del server. Questo schema offre una protezione contro gli attacchi replay permettendo al server di verificare l'attualità del nonce, che sarà inserito negli header *Authorization* delle richieste successive.

d. Opaque: è un campo utilizzato solo dal server, in cui può inserire ciò che vuole per scopi sconosciuti al client, che dovrà semplicemente ricopiare nei messaggi di richiesta successivi. Spesso è utilizzato per memorizzare informazioni di stato relative alla sessione di autenticazione.

e. Algorithm: può essere "MD5" o "MD5-sess" e determina la struttura di un termine dell'equazione per il calcolo del response. Se viene utilizzato l'algoritmo "MD5-sess", successivamente ad un'autenticazione con esito positivo viene generata e condivisa una "session key", il cui calcolo includerà i valori nonce iniziali del server e del client. La "session key" permette al server di accettare dei response contenenti richieste consecutive del client, anche senza rigenerare un nonce "fresco" ogni volta. In questo modo si riduce il numero di messaggi scambiati permettendo al client di calcolare e fornire i response autonomamente, senza dover essere sfidato ogni volta da un challenge. Una "session key" è valida per tutta la durata di una sessione di autenticazione, che finisce quando il server invia al client un nuovo header "WWWAuthenticate" o "Authentication-Info". Se il parametro *Algorithm* è assente, si assume sia MD5.

f. Stale: se "true", indica al client che la sua precedente richiesta è stata rifiutata perché il nonce era scaduto, ma il response era stato calcolato con la giusta password. In questo caso il client userà la stessa password, ma con il nuovo server nonce, per ricalcolare il response. Se "false", allora deve essere richiesto all'utente del client di fornire nuovamente la password.

3. Response: l'UAC calcola il response e rispedisce la richiesta, questa volta includendo l'header *Authorization*, che conterrà i seguenti parametri:

a. Username: il nome utente del “realm” specificato.

b. Realm: lo stesso dell’header *WWW-Authenticate*.

c. Nonce: lo stesso dell’header *WWW-Authenticate*.

d. Digest-URI (uri): il valore del Request-URI.

e. Qualità della protezione (qop): indica la qualità della protezione scelta dall’UAC.

f. Nonce count (nc): indica il numero di request distinte inviate dall’UAC usando lo stesso valore di nonce del messaggio attuale. Il nonce count è utilizzato come input nel calcolo del response, permettendo al server di rilevare dei replay mantenendo la sua copia di nonce count.

g. Client nonce (cnonce): questo parametro deve essere presente se è stato specificato un parametro qop. E’ il nonce generato dall’UAC e utilizzato nel calcolo e nella validazione del response. Il suo scopo è di proteggere la password contro gli attacchi di tipo “chosen plaintext”. Un attacco di questo tipo può essere lanciato in questo caso da un malintenzionato che si finge server SIP e sceglie i valori di nonce. Il client nonce permette all’UAC di introdurre una maggiore entropia nell’output, controllata solo dal client.

h. Response: il response dell’UAC.

i. Opaque: lo stesso dell’header *WWW-Authenticate*.

j. Algorithm: lo stesso dell’header *WWW-Authenticate*.

Descriveremo di seguito la procedura utilizzata dal client per calcolare il response. Viene utilizzata la seguente notazione:

- KD (secre, data) = H(secret || ":" || data), dove H denota l'applicazione della funzione di hash MD5 ai parametri in input
- unq(param), che denota il valore senza virgolette del parametro “param”.

Successivamente sarà specificato come calcolare A1 e A2.

Quando la qualità della protezione (qop) è impostata ad “auth” o “auth-int”, il response è calcolato come:

response = KD(H(A1), unq(nonce) || ":" || nc || ":" || unq(cnonce) || ":" || unq(qop) || ":" || H(A2))

quando invece il parametro qop è assente, si usa:

$$response = KD(H(A1), unq(nonce) \,||\, ":" \,||\, H(A2))$$

Se il parametro del campo “algorithm” è “MD5” o non è specificato nessun algoritmo, il termine A1 è calcolato come:

$$A1 = unq\ (username) \,||\, ":" \,||\, unq(realm) \,||\, ":" \,||\, passwd$$

dove “passwd” denota la password dell’utente. Quando viene usato l’algoritmo “MD5-sess”, la formula per A1 diventa:

$$A1 = H(unq(username) \,||\, ":" \,||\, unq(realm) \,||\, ":" \,||\, passwd \,||\, ":" \,||\, unq(nonce) \,||\, ":" \,||\, unq(cnonce))$$

Si può osservare come il calcolo del response includa un valore hash calcolato sulla password dell’utente (insieme agli altri parametri) ma non la password esplicita.

Quando la qop è “auth” o non è specificata, il termine A2 si calcola come:

$$A2 = method \,||\, ":" \,||\, URI$$

dove “method” denota il nome del metodo SIP. Se è selezionato “auth-int” invece la formula diventa:

$$A2 = method \,||\, ":" \,||\, URI \,||\, ":" \,||\, H(body)$$

Dove “body” denota il corpo della richiesta SIP, di cui si protegge in questo modo l’integrità.

4. Autenticazione mutuale: assumendo che l’utente si sia autenticato con successo, il server può includere un header *Authentication-Info*, il cui principale scopo, nel contesto SIP, è quello di fornire un meccanismo di autenticazione mutuale. L’header *Authentication-Info* contiene i seguenti parametri:

    a. Nextnonce: fornisce il valore nonce che l’UAC deve usare per autenticare la richiesta successiva.

    b. Quality of protection (qop): indica la qualità della protezione fornita dal server al messaggio di response.

    c. Client nonce (cnonce): uguale al campo cnonce dell’header *Authorization* inviato dal client.

    d. Nonce count (nc): la copia del server del conto dei nonce.

    e. Response authentication (rspauth): un response calcolato dal server per provare che il server conosca la password dell’utente. Vengono utilizzate le

stesso equazioni del client, eccetto per il termine A2, che ha una struttura differente. Quando la qualità di protezione (qop) è impostata ad “auth” o non è presente, il termine A2 del server response authentication è calcolato come:

$$A2 = ":" \,||\, URI$$

Se invece è selezionato “auth-int”, la formula diventa:

$$A2 = ":" \,||\, URI \,||\, ":" \,||\, H(body)$$

dove “body” denota il corpo del response SIP, di cui viene garantita in questo modo l’integrità.

3.5 Il protocollo di Key Agreement

I protocolli finora presentati permettono una fase di autenticazione, che rappresenta solo il primo passo per garantire un sistema sicuro. Infatti, una volta accertata l'identità delle parti è necessario creare un canale sicuro su cui trasmettere i propri dati garantendo una piena confidenzialità ed integrità degli stessi. Perchè l'instaurazione di questo canale sicuro avvenga, è necessario l'uso di un sistema in grado di applicare la crittografia anche ai pacchetti che viaggiano sul canale di comunicazione, una crittografia alla cui base sta un segreto condiviso tra le parti e sconosciuto a terzi. I protocolli di Key-Agreement permettono ai partecipanti di una comunicazione di accordarsi su questo segreto (nello specifico una chiave segreta) per poi, successivamente, garantire l'instaurazione di un canale sicuro.

L'accordarsi, da parte di due entità, su una chiave segreta, richiede molto spesso una fase successiva di scambio di essa, che, in base al protocollo scelto, può avvenire o su un canale sicuro oppure su un canale insicuro (pubblico).

Il protocollo Diffie-Hellman (14), per l'accordo e lo scambio di chiavi segrete, utilizza come canale di trasmissione un canale insicuro, senza la necessità che le parti si siano scambiate informazioni o si siano incontrate in procedenza.

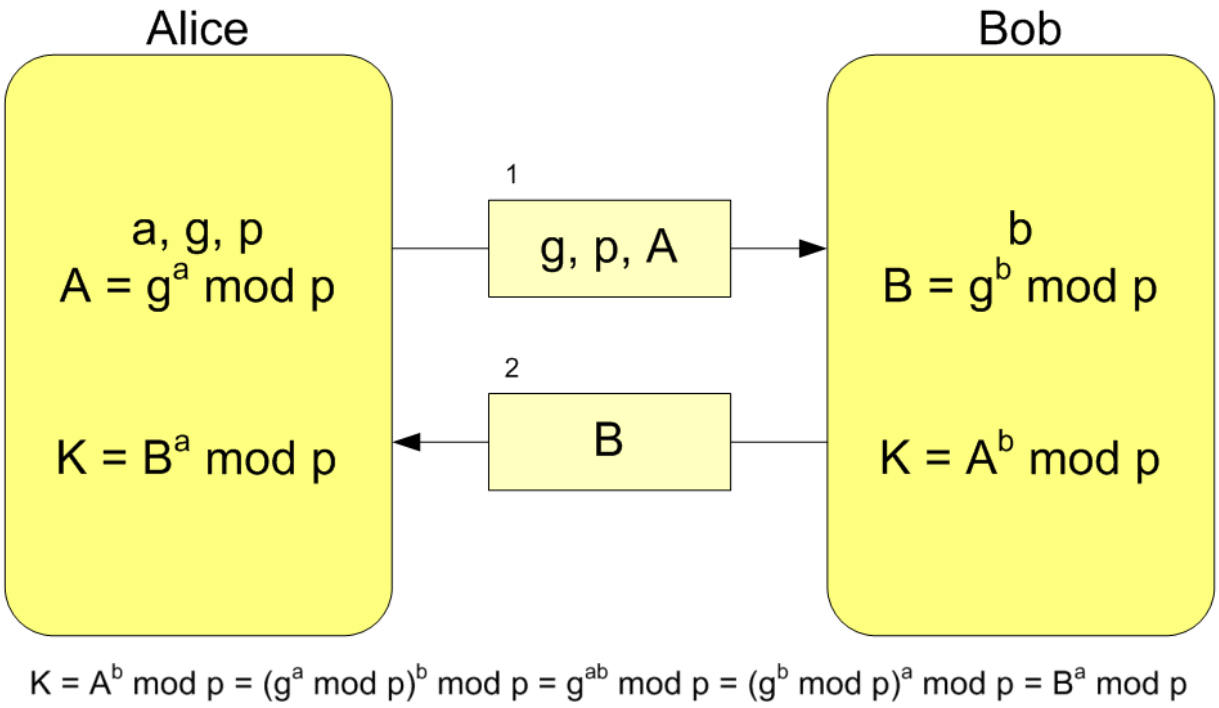

Fig. 3.2 Scenario Diffie-Hellman Key Exchange

In figura 3.2 è presente uno scenario tipico che raffigura uno scambio di chiavi segrete tramite il protocollo Diffie-Hellman. Due interlocutori Alice e Bob voglione eseguire uno scambio di chiavi:

1) Viene scelto un numero casuale g, detto generatore o radice primitiva modulo p, dove p rappresenta un numero primo.
2) Alice sceglie un numero casuale a e calcola il valore $A = g^a \bmod p$, inviandolo, insieme ai valori g e p, a Bob.
3) Bob sceglie un numero casuale b e anch'esso calcola $B = g^b \bmod p$ e lo invia ad Alice.
4) A questo punto Alice calcola $K_a = B^a \bmod p$, mentre Bob calcola $K_b = A^b \bmod p$
5) Finite le fasi di calcolo entrambi gli interlocutori sono in possesso della chiave segreta.

Lo scenario Diffie-Hellman, qui presentato, soffre di un debolezza. Se le parti interessate non condividono un segreto, nello specifico una chiave, l'algoritmo in questione può essere sottoposto ad attacchi MITM. Le informazioni pubbliche scambiate tra i partecipanti durante l'esecuzione dell'algoritmo possono essere intrecettate e modificate. E' evidente, quindi, la necessità di un qualche tipo di autenticazione in grado di garantire l'identità delle parti, che si basi appunto sull'uso del segreto pre-condiviso (message authentication).

Con il termine PSK (Pre Shared Key) si intende appunto questo segreto pre-condiviso ottenuto tramite un canale sicuro. Le caratteristiche di questa chiave sono strettamente legate al sistema che la utilizza; alcuni progetti richiedono che tale chiave sia in un formato particolare: una parola d'ordine specifica, una frase oppure una stringa esadecimale. In ogni caso ogni partecipante alla comunicazione deve essere a conoscienza del segreto.

3.6 Il protocollo ZRTP

Una particolare attenzione va al protocollo di key-agreement ZRTP (15), in quanto utilizzato per l'integrazione di uno scambio di chiavi all'interno del sistema ABPS. ZRTP è un protocollo che non si basa su alcun tipo di protocollo di segnalazione (SIP, H.323), che utilizza l'algoritmo Diffie-Hellman per effettuare lo scambio di segreti tra partecipanti, che necessità che entrambe le parti partecipanti alla negoziazione delle chiavi supportino ZRTP ed é di piena integrazione con il protocollo SRTP per garantire crittografia in appoggio a RTP. ZRTP è un protocollo di key-agreement con alcune particolarità:

- Essendo basato su uno scambio di chiavi Diffie-Hellman, non necessita di alcun tipo di infrastruttura intermediaria (PKI) o autorità di certificazione per eseguire uno scambio di chiavi in maniera sicura ma garantisce la propria sicurezza, atta a scongiurare attacchi di tipi MITM, sull'utilizzo di segreti condivisi: l'uso di una PSK (Pre Shared Key) oppure di SAS(Short Authenticazion String).
- L'utilizzo delle stesse porte di comunicazione di RTP diminuisce le problematiche di

trasmissione di dati.

- garantire chiavi forward secrety, ovvero, in grado di essere eliminate al termine della sessione corrente RTP evitando anche attacchi dovuti a decriptazioni di comunicazioni audio e/o video precedentemente registrate
- esclude completamente terze parti, compresi gli operatori di rete, garantendo una maggior sicurezza di cui gli unici responsabili sono esclusivamente i partecipanti alla comunicazione
- esclude completamente qualunque tipo di trapdoors; sia non maligni, voluti per esigenze di controllo da gli stessi progettisti di ZRTP, sia maligni, voluti da malintenzionati.

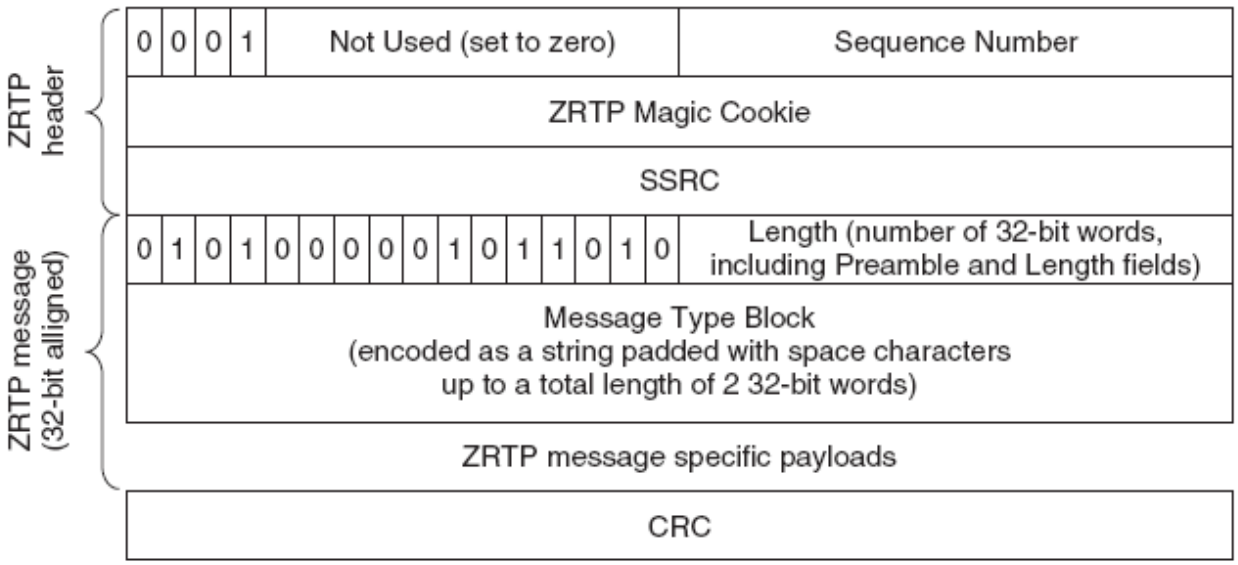

Fig. 3.3 Pacchetto ZRTP

La figura 3.3 rappresenta la struttura di un pacchetto ZRTP. L'header ZRTP utilizza un formato che è in parte comune a quello RTP. Esso specifica inoltre:

- Un numero di sequenza che viene inizializzato ad un valore casuale ed incrementato ogni volta che viene spedito un pacchetto ZRTP. Il numero di sequenza viene usato per stimare il numero di pacchetti persi o arrivati fuori ordine.

- Un valore SSRC del flusso RTP con il quale ZRTP condivide gli endpoint.

Un pacchetto ZRTP trasporta un messaggio ZRTP, che comincia con un preambolo, seguito dalla lunghezza del messaggio, il tipo di messaggio e i payload. Il pacchetto termina con un CRC di 32 bit per rilevare errori nello schema di trasmissione, come meccanismo supplementare al checksum UDP.

Capitolo 4

ABPS

4.1 La problematica della Seamless Mobility

Con il termine Seamless Mobility si intende un insieme di architetture per l'integrazione di reti eterogenee, responsabili di indetificare in maniera univoca un nodo multi-homed, rendere possibile ad ogni nodo di essere raggiunto da altri nodi, con cui ha già effetuato una precedente connessione, e monitorare i parametri QoS permettendo, in caso di handoff, la selezione della rete migliore. Prendendo una comunicazione IP-based, nello specifico uno scenario VoIP, l'indirizzo IP rappresenta un'identificatore univoco per un host. Essendo tale host di tipo mobile, la sua capacità di poter cambiare interfaccia di rete all'occasione, vanifica la possibilità di identificazione tramite IP, in quanto ogni interfaccia di rete di cui dispone il dispositivo è associata ad un indirizzo IP differente. Ciò rappresenta un problema, specialmente per altri host mobile che non potranno raggiungere l'host in questione prima di essere a conoscenza del suo nuovo indirizzo IP, con una conseguente discontinuità all'interno della comunicazione. La soluzione comune a tutte le architetture di Mobile Management (MMA), anche se implementate in strati ISO/OSI differenti, si basa su due semplici principi:

1. Definire un identificatore univoco per il terminale mobile, indipendente alla provenienza dell’host.
2. Fornire un servizio di localizzazione,sempre raggiungibile da altri host mobile, per mantenere un’associazione tra l’identificatore univoco dell'host mobile ed il suo reale indirizzo di provenienza.

Le architetture dello strato di rete attualmente utilizzate per la seamless mobility sono le seguenti:

- Mobile IP versione 6 (MIP)
- Fast Handover Mobile IPv6 (FMIP), un'ottimizzazione della precedente
- Hierarchical Mobile IPv6 (HMIP)
- Proxy Mobile IPv6 (PMIP)

Tutte le architetture elencate aggiungono all'interno della rete alla quale appartiene l'host mobile un'entità, detta Home Agent, avente funzione di Location Registry. Queste architetture, essendo basate su IPv6, richiedono che entrambi gli end, l'host mobile e gli altri host che vogliono raggiungerlo, abbiano capacità IPv6 e necessitano di infrastrutture di rete che supportino IPv6. Inoltre le specifiche MIPv6 non permettono l'utilizzo simultaneo di più NIC dell'host mobile. Per ogni terminale può essere registrato l'indirizzo di un solo NIC nell'home agent, lasciando di fatto irrisolto il problema degli intertechnology handoff. In più, la latenza dell'handover risulta molto alta a causa di numerosi messaggi di autenticazione.

Per le architetture situate tra il livello di rete e il livello di trasporto, come il protocollo Host Identity Protocol (HIP) e il Location Independent Addressing for IPv6 (LIN6), il location registry è rappresentato da una funzione di mapping simile al DNS che opera come servizio esterno alla rete di provenienza dei nodi. L'approccio di queste architetture si basa sull'aggiunta, in tutti gli host mobile comunicanti nella rete, di uno strato intermedio tra il livello di rete e quello di trasporto ed è proprio questo principio a renderlo non conveniente, in quanto se può essere ragionevole una modifica simile nel sistema mobile di interesse, non lo è negli altri host partecipanti, che potrebbero anche essere dei nodi fissi non interessati a supportare la mobilità dell'host mobile.

Per il livello di trasporto l'approccio comune è molto differente. Le architetture per questo livello, come il Datagram Congestion Control Protocol (DCCP), il Mobile Stream Control Trasport Protocol (m-SCTP) e l'estensione TCP-migrate, integrano le fuzioni di location registry all'interno dei terminali stessi, che hanno il compito di informare gli altri end a ogni cambio di indirizzo IP. Con questo approccio il problema della localizzazione resta se i due (o più) end perdono la loro configurazione IP nello stesso momento, diventando così reciprocamente irraggiungibili.

Infine, per le architetture a livello di sessione, come il protocollo Terminal Mobility Support Protocol (TMSP), abbiamo che il principio base sta nell'utilizzo di un server SIP ausiliario,

esterno alla rete di accesso, che ha la funzione di location registry per mappare alle URI degli utenti il loro indirizzo IP corrente. Ogni host mobile utilizza il protocollo SIP per inviare un messaggio REGISTER al server SIP per aggiornare la sua localizzazione. Quest'ultima soluzione non risulta efficiente perché quando avviene una riconfigurazione nell'IP dell'host mobile, il tempo durante il quale tale host comunica al server di aggiornare il proprio record, mediante un handshake con diverse fasi, introduce un delay inaccettabile, durante il quale l'host interrompe la comunicazione con il terminale a cui è in quel momento connesso, che non può continuare la trasmissione prima dell'arrivo del messaggio di signaling da parte del server.

4.2 La soluzione ABPS

ABPS (16) è un ottima soluzione ai problemi sopra elencati in uno scenario multihoming su un terminal mobile VoIP wireless, un estensione per i protocolli RTP e SIP in grado di rispettare a pieno i requisiti QoS. Le pricipali componenti sono:

1. Sip mobility: entità a livello applicazione in grado di gestire un handover layer-3, ovvero la riconfigurazione di un indirizzo IP.
2. Vertical mobility: entità a livello data-link e network avente il compito di monitorare lo stato di tutte le interfacce di rete su un dispositivo, gestendo, sulla base di opportune metriche e politiche, la selezione dell'interfaccia di trasmissione.

Le due entità realizzate sono rispettivamente poste: una su un server collegato alla rete pubblica (vertical mobility), l'altra posta su un terminale mobile (Sip mobility).

L'applicazione server rappresenta il nodo centrale di una comunicazione VoIP; ogni terminal mobile che vuole effettuare una comunicazione dovrà passare dal server, il quale gestirà in maniera trasparente le fasi cruciali per l'instaurazione, l'uso e la chiusura della sessione. Il tutto amministrando in maniera opportuna i protocolli SIP ed RTP/SRTP, incluse le varie estensioni ad essi collegate, come appunto ZRTP. Il terminale mobile che effettua un collegamento al server è all'oscuro di questo passaggio, infatti, nella connessione stabilita, pensa di aver trovato un collegamento diretto all'altro terminale con cui vuole comunicare.

Il terminale è all'oscuro del collegamento con il server, pensando di aver trovato un collegamento diretto, solamente dopo le giuste fasi di autenticazione il server avrà il compito di ridirigere il flusso verso il terminale chiamato, similmente a quanto fatto da un home agent in MobileIP.

L'applicazione presente sul sistema mobile è dotata di un architettura a tre livelli:

1. Primo Livello: costituisce il livello più basso dell'architettura che permette la realizzazione di handover e loadbalacing.
2. Secondo Livello: costituito da un insieme di strutture e funzioni necessarie per garantire autenticazione mutuale e firma dei pacchetti.
3. Terzo Livello: livello che rappresenta il vero e proprio client SIP
4. Quarto Livello: l'interfaccia GUI, che garantisce l'iterazione tra utente finale e terminal mobile di tutte le funzionalità implementate.

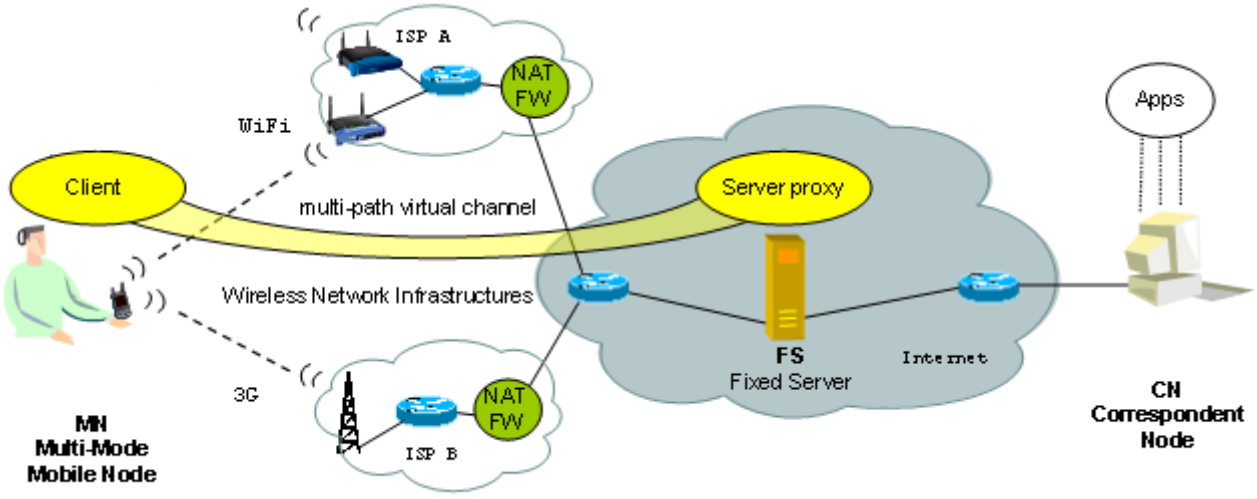

Fig. 4.1 Scenario ABPS

La figura 4.1 presenta uno scenario multihoming handover in cui è presente ABPS. Le due parti introdotte dall'architettura in questione sono:

- Un Client ABPS per il nodo mobile, eseguito direttamente sul terminale, svolge un ruole simile al foreign agent, con la differenza che non viene gestito dalla rete di accesso ma dallo stesso terminale
- Un Server Proxy ABPS che si occupa di gestire la mobilità in piena collaborazione con il client.

La soluzione che propone ABPS è basata sull'utilizzare un sistema VoIP wireless basato su SIP non IP-centrico, in grado di distinguere il traffico proveniente da un certo client registrato senza potersi affidare all'indirizzo IP da cui questo proviene. Questo perché il client sul terminale mobile deve poter trasmettere i suoi dati da un qualunque indirizzo IP, anche da più indirizzi IP contemporaneamente, in un qualunque momento della sessione, senza dover avvisare il server della riconfigurazione. Questo approccio fa sì che non sia necessario un protocollo di sincronizzazione, eliminando i relativi ritardi di trasmissione e mantenendo la continuità.

4.3 Estensione dei protocolli SIP e RTP in ABPS

Anche se collaboranti, è stato necessario implementare due diverse soluzioni, una per SIP ed una per RTP, in quanto i protocolli richiedevano esigenze differenti. In particolare un flusso multimediale come RTP, a differenza di SIP, è composto dalla trasmissione di un gran numero di datagrammi di piccole dimensioni, quindi, si necessita di un sistema che garantisca la "produzione" di datagrammi di dimensioni standard e algoritmi di cifratura e decifratura che non carichino eccessivamente la CPU, specialmente se il sistema opera su un dispositivo mobile quale uno smart-phone.

4.3.1 ABPS-SIP

L'ABPS SIP è un estensione di SIP, che stabilisce un canale sicuro tra le due estremità del percorso multi-channel virtuale (client e server ABPS) e l'autenticazione dei messaggi SIP utilizzando un regime di tipo digest. Tutti i client interessati ed il server condividono una PSK (pre shared key) che viene stabilita durante la fase di configurazione off-line e mai trasmessa. Per ogni messaggio SIP da inviare questa chiave è utilizzata, oltre che nelle funzioni HMAC, per generare un impronta (fingerprint) di quel messaggio. Inoltre un altro ruolo importante della PSK è quello di fungere da generatrice di una chiave di sessione per la crittografia dei pacchetti streaming (RTP/SRTP).

In figura 4.2 è rappresentata la fase iniziale di registrazione, dove, tramite uno scambio di messaggi SIP, un terminal mobile si registra ed autentica al sistema ABPS per poi successivamente autenticarsi, sempre passando attraverso il server, presso il proprio gestore di servizio VoIP.

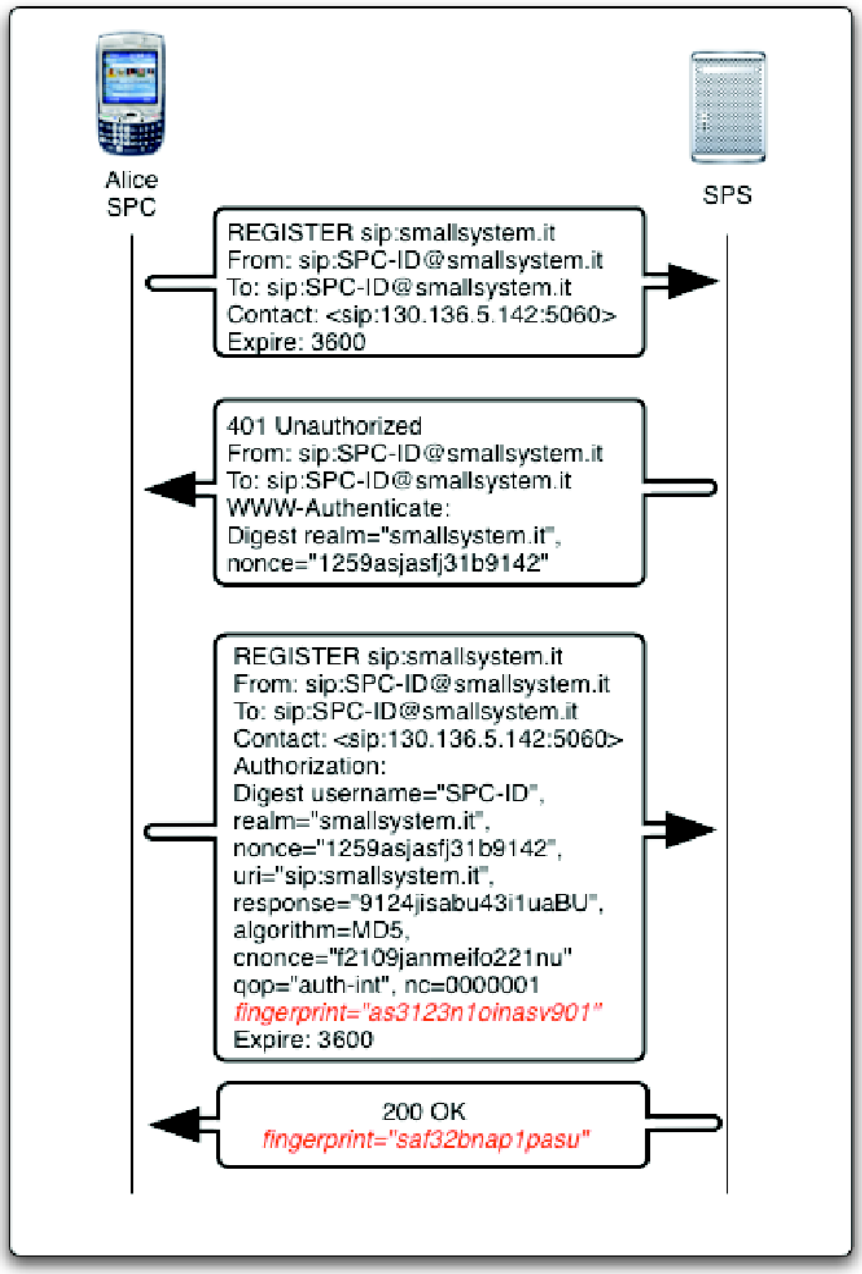

Fig. 4.2 Four-way handshake nell'autenticazione ABPS

Le fasi raffigurate sono:

PRIMA FASE (eseguita da terminale mobile):

1. Il terminale mobile all'avvio del client genera un messaggio di REGISTER semplice, senza credenziali per l'autenticazione. Nel campo *From* e *To* specifica il suo identificatore univoco.
2. Invia il messaggio al server.

SECONDA FASE (eseguita da server):

1. Dopo aver ricevuto il REGISTER, il server genera un messaggio di errore per segnalare al client che il servizio richiesto necessita di autorizzazione. In questo caso, trattandosi di un server proxy, il messaggio inviato sarà un *407 Unauthorized*. Dopo aver generato un nonce, generalmente casuale, include il suo *realm*, il tipo di autenticazione (Digest, plaintext, Extended AKA) che necessita per il response e la *quality of protection* (qop).
2. Calcola l'hash del messaggio che sta inviando e lo salva in memoria per la fase 4.
3. Invia il messaggio al terminale mobile.

TERZA FASE (eseguita dal terminale mobile):

1. Il terminale mobile calcola l'hash del messaggio di challenge ricevuto (quello generato nella fase 2) e lo salva in memoria per un utilizzo successivo all'autenticazione. hash_challenge=SHA-1-160(msg_challenge)
2. Calcola il response secondo le stesse modalità e procedure dell'autenticazione HTTP, utilizzando come password la master_key.
3. Aggiunge l'header di autenticazione, con il response appena calcolato, al messaggio generato in fase 1.
4. Calcola l'hash del messaggio, comprensivo di response: hash_response=SHA-1-160(msg_response)
5. Calcola una chiave temporanea, basata sulla master_key e su alcune credenziali del messaggio di response: masq_master_key=SHA-1-160("20:"||master_key||":"||

cnonce||":"||nonce)

6. Calcola il fingerprint, utilizzando la funzione crittografica HMAC: fingerprint=HMAC-SHA-1-160(msg_response,masq_master_key)
7. Aggiunge l'header fingerprint in coda a msg_response e invia il messaggio completo al server.

QUARTA FASE (eseguita dal server)

1. Il server rimuove dal messaggio ricevuto l'header fingerprint e lo salva in memoria.
2. Calcola la chiave temporanea basata sulla master_key, utilizzando nonce e cnonce del messaggio ricevuto: masq_master_key=SHA-1-160("20:"||master_key||":"|| cnonce||":"||nonce)
3. Calcola il fingerprint. fingerprint=HMAC-SHA-1 -160(msg_response,masq_master_key).
4. Confronta il fingerprint calcolato con il fingerprint memorizzato al punto 1.
5. Se la verifica è andata a buon fine, crea un nuovo messaggio di acknowledgment *200 OK*
6. Calcola la chiave di sessione per l'autenticazione di integrità dei messaggi: sess_ia_key=HMAC-SHA-1-160((hash_response||":"||hash_challenge||0x00) , masq_master_key)
7. Calcola il fingerprint del messaggio *200 OK* utilizzando la chiave di sessione appena costruita: fingerprint=HMAC-SHA-1-160(msg_200_OK , sess_ia_key)
8. Aggiunge l'header fingerprint al messaggio *200 OK*
9. Invia il messaggio all'MN.

QUINTA FASE(eseguita dal terminal mobile):

1. Rimuove l'header fingerprint dal messaggio *200 OK* ricevuto e lo salva in memoria.
2. Calcola la chiave di sessione (qui torna utile *hash_challenge*, memorizzato in fase 3): sess_ia_key=HMAC-SHA-1-160((hash_response||":"||hash_challenge||0x00) , masq_master_key)
3. Calcola il fingerprint del messaggio: fingerprint=HMAC-SHA-1-160(msg_200_OK ,

sess_ia_key)

4. Se il fingerprint salvato al punto 1 coincide con il fingerprint appena calcolato, il terminal mobile termina l'autenticazione con esito positivo e procede con l'esecuzione del client, altrimenti ripete l'autenticazione oppure termina.

Effettuata la registrazione nei successivi messaggi SIP (INVITE, BYE, ecc...) si renderà necessario l'aggiungere di un header di 36 byte al messaggio SIP cosi composto (figura 4.3):

- UID: un campo che identifica l'identità del mittente.
- SeqNUM: un numero progressivo utilizzato per evitare replay-attach.
- Fingerprint: calcolato su tutto il messaggio garantisce al ricevitore di verificare l'integrità del messaggio: fingerprint=HMAC-SHA-1(msg_sip,session_ia_key).

UID	SeqNUM	Fingerprint
8(20)Byte	8(10)byte	20(40)byte

Fig. 4.3 Header aggiunto al pachetto SIP per garantire autenticità

4.3.2 ABPS-RTP

L'ABPS-RTP è stato progettato come un insieme di protocolli standard a livello di applicazione, che collaborano per fornire un canale sicuro RTP tra due sistemi finali. In particolare, ABPS-RTP fornisce l'autenticazione su base flusso RTP, indipendentemente dall'indirizzo IP del mittente, l'integrità e, facoltativamente, riservatezza. In altre parole, il ricevitore distingue tra i diversi flussi RTP ed identifica il mittente di un messaggio, anche quando cambia il proprio indirizzo IP. Così ABPS-RTP permette di passare dinamicamente messaggi RTP attraverso tutti i percorsi disponibili tra il client e il server ABPS. Per recapitare i messaggi di un determinato flusso multimediale, ABPS-RTP utilizza una

estensione standard, il Secure Real-time Transport Protocol (SRTP). SRTP aggiunge un header con un tag di autenticazione calcolato utilizzando una chiave di cifratura che viene creata per tale flusso RTP. Così, prima di creare un flusso RTP protetto tra due sistemi terminali, SRTP richiede una fase preliminare in cui un protocollo di key-agreement genera una chiave di cifratura. SRTP non specifica un particolare protocollo di key-agreement tra quelle disponibili, come Mikey, SDES, DTLS, ZRTP, tuttavia, grazie alle sue prestazioni, si sceglie il protocollo di intesa denominato key-ZRTP.

Le fasi principali per instaurare un canale RTP sicuro, passano dall'utilizzo iniziale di ZRTP per poi arrivare alla fase lavorativa di SRTP, seguito dall'instaurazione del flusso RTP. E' importante ricordare che in tutte queste fasi l'autenticazione dei messaggi SIP scambiati tra client e server, per quanto riguarda la prima fase, è sempre garantita tramite l'utilizzo e l'aggiunta nei pacchetti scambiati di un UID, un SeqNum ed il fingerprint (autenticazione SIP-ABPS):

PRIMA FASE:

Rappresenta la fase in cui il Client effettua una richiesta di chiamata tramite l'invio di un messaggio SIP di INVITE. L'autenticazione viene garantinta da ZRTP usufruendo del segreto pre-condiviso tra client e server (master-key).

1) Il client genera un messaggio di Hello-ZRTP e lo salva in memoria.
2) Calcola l'hash del messaggio Hello-ZRTP.
3) Prima di inviare l'INVITE SIP, aggiunge nel corpo SDP di tale pacchetto un attributo zrtp-hash contenente l'hash del messaggio hello-ZRTP.
4) Il client invia la richiesta di INVITE al server.
5) Una volta ricevuto il messaggio di INVITE, il server crea anch'esso un messaggio di Hello-ZRTP, lo salva in memoria, genera l'hash di Hello-ZRTP, aggiunge l'hash ed invia il pacchetto al client.

SECONDA FASE:

La sessione di scambio INVITE SIP è terminata. E' ora possibile instaurare una sessione multimediale RTP/SRTP necessaria a completare le fasi di autenticazione, scambio di chiavi

ed infine la comunicazione vera e propria.

6) Ricevuto il messaggio dal server, il client invia il messaggio di Hello-ZRTP al server utilizzando la porta assgnata al traffico RTP (in questa fase si è ufficialmente passati da un canale SIP ad uno RTP), il quale ne calcola l'hash e lo confronta con l'hash prelevato dall'header del messaggio precedente di INVITE. Successivamente la stessa operazione viene eseguita invertendo le parti.
7) Se il reciproco confronto dei messaggi di hello-ZRTP avviene con successo, è possibile passare alla fase successiva in cui avverrà lo scambio di chiavi di sessione.

SECONDA FASE:

Nella seconda fase avviene la generazione di chiavi private di sessione ZRTP tramite l'agoritmo di key-agreement Diffie-Hellman. Tali chiavi hanno lo scopo principale di generare le successive chiavi di sessione.

TERZA FASE:

Sia il client sia il server sono in possesso di chiavi di sessione in grado di essere utilizzate da SRTP per crittografare i pacchetti audio da inviare. La comunicazione RTP può avvenire in piena sicurezza.

In figura 4.4 è rappresentata la fase inerente a ZRTP. Questo scenario parte dalla fase successiva allo scambio di messaggi di INVITE (contenenti l'hash dell'hello-zrtp), ovvero, la sessione SIP è terminata e viene instaurata quella RTP/SRTP.

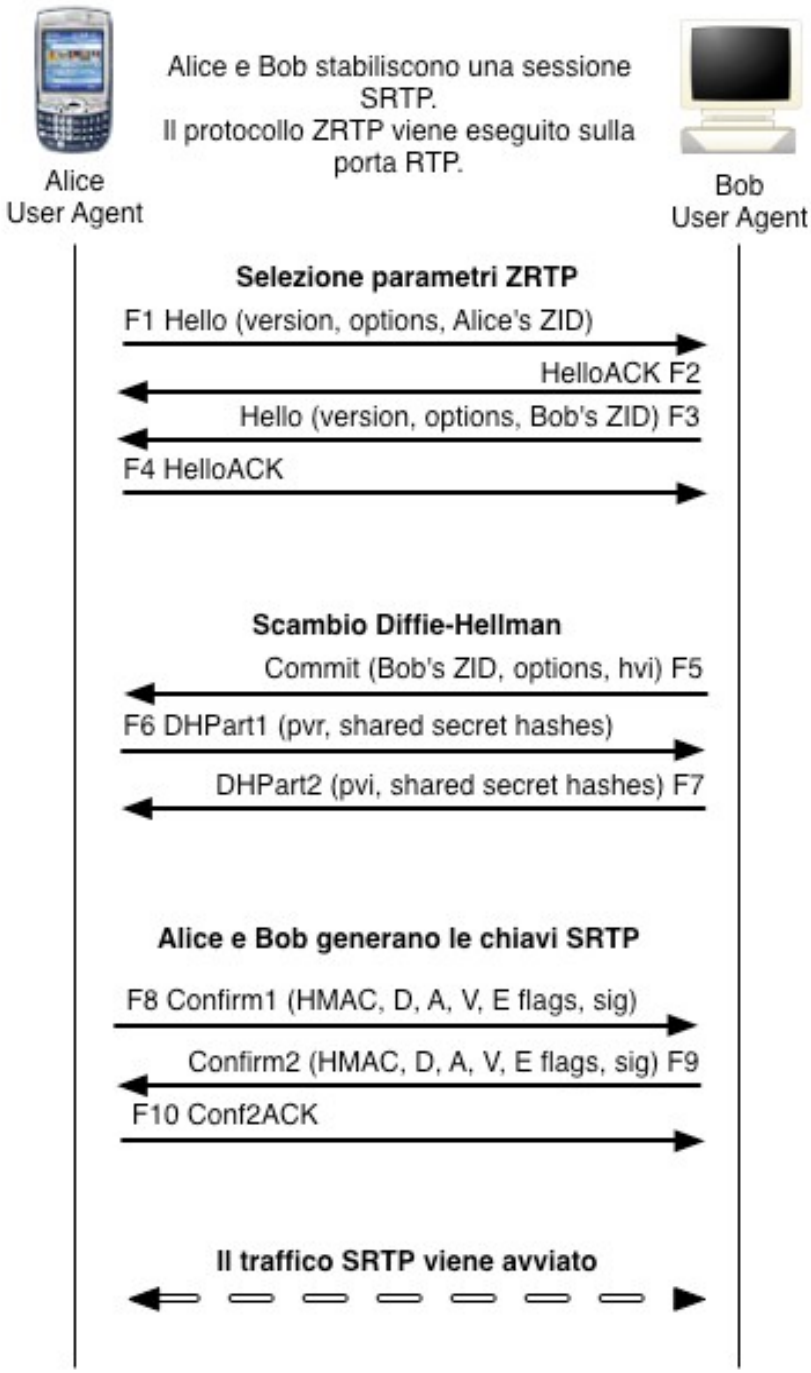

Fig. 4.4 Scambio messaggi ZRTP per l'instaurazione di una sessione SRTP

Capitolo 5

Progettazione

5.1 SYMBIAN OS

Nel corso degli ultimi decenni i dispositivi mobili, PDA (personal digital assistant) e smartphone, hanno conquistato fette di mercato sempre più rilevanti. Le funzionalità di cui nel corso degli anni si sono arricchiti erano fino a poco tempo fa impensabili. Con lo sviluppo di hardware sempre più sofisticato, è stato possibile fornire tali dispositivi di caratteristiche sempre più complesse. Con l'aumento delle potenzialità legate alle capacità di calcolo dei dispositivi mobili, i produttori hanno dovuto accrescere i propri sforzi nello sviluppo di sistemi operativi, in modo da presentare soluzioni software sempre più efficienti e complete. Attualmente i sistemi operativi per dispositivi mobili maggiormente diffusi sono:

- *Symbian*
- *Rim*
- *Windows Mobile*
- *iPhone OS*
- *Mobile Linux*
- *Android*
- *Palm OS*

I sistemi operativi più usati per gli smartphone sono Symbian OS *(*Symbian Foundation), Palm OS (sviluppato dalla PalmSource), Windows CE (Microsoft), Windows Mobile (Microsoft), BREW (Qualcomm), Linux, Android e iPhone OS. Symbian è oggi il più diffuso e, malgrado nell'ultimo periodo inizi a soffrire a causa della forte concorrenza imposta da nuovi competitor, quali apple e google, rimane ancora il sistema operativo più venduto.

Symbian è un sistema operativo disegnato per fronteggiare le problematiche relative alle caratteristiche hardware embedded degli smartphone, molto diverse per alcuni aspetti da quelle dei calcolatori, in particolare in termini di consumo di potenza e quantità di memoria disponibile, oltre ad un pieno supporto per i requisiti specifici del trasporto dati. Tassello fondamentale per i dispositivi mobile di nuova generazione (2G e 3G).

Le caratteristiche fondamentali di Symbian (17) sono:

Performance: progettato per massimizzare la durata della batteria attraverso un power management specifico, in base alle caratteristiche del dispositivo.

Multitasking: Telefonia, messaggistica e comunicazioni sono componenti fondamentali. Tutte le applicazioni sono progettate per funzionare in parallelo.

Standard: L'uso di tecnologie basate sugli standard di settore è un principio fondamentale di Symbian OS, ciò assicura l'interoperabilità delle applicazioni sviluppate.

Software object-oriented: Symbian OS nasce con una struttura object-oriented e ha un'architettura altamente modulare.

Gestione ottimizzata della memoria: Gli eseguibili per questo sistema operativo hanno una dimensione decisamente ridotta e i requisiti di runtime sono minimizzati.

Meccanismi di sicurezza: sono presenti funzionalità al fine di consentire comunicazioni sicure e l'archiviazione sicura dei dati.

Internazionalizzazione: il sistema ha incorporata la gestione del set di caratteri Unicode per garantire un semplice sviluppo delle funzionalità di localizzazione.

A Giugno del 2008 la Nokia annuncia l'acquisto completo di tutte le azione di Symbian ed insieme ad altre società (Motorola, Sony Ericsson, NTT DoCoMo, Texas Instruments, Vodafone, Samsung, LG e AT&T) dà vita alla Symbian Foudation, un'organizzazione non-profit che gestirà Symbian OS, rilasciando la piattaforma sotto licenza EPL (Eclipse Public

License), una licenza libera. Questo radicale cambiamento ha permesso una sempre più fiorente crescita di una comunity di sviluppatori in tutto il mondo.
Inoltra il passaggio da sistema proprietario a sistema completamente Free, ha portato anche un cambiamento per quanto riguarda l'ambiente di sviluppo IDE (Integrated Development Environment), passando da un ambiente CodeWarrior a Carbide.c++ () che tuttora detiene il primato come primo ambiente di sviluppo per applicazioni su Symbian.
Symbian OS è utilizzato in molteplici smartphone, ad esempio i Nokia Series 60, Series 80 e Series 90, e UIQ (la piattaforma di UIQ Technology). Ogni piattaforma ha bisogno di un SDK (Software Development Kit) specifico ed attualmente Symbian Foundation non ha rilasciato una sua versione dell'SDK, cosa che farà "as soon as possible". Per il momento non è ancora disponibile una versione open dell'SDK, e per lo sviluppo bisogna appoggiarsi all'SDK realizzato da NOKIA. Quest'ultimo contiene le librerie e gli "header" files necessari per sviluppare applicazioni per Symbian OS, e un emulatore che consente di effettuare dei test utilizzando un normale pc. L'SDK della nokia deve essere usato in concomitanza con l'Application Development Toolkit (ADT).
L'SDK da l'accesso alle API pubbliche, e lavorando esclusivamente su quest'ultime si ha la sicurezza che le applicazioni sviluppate funzioneranno su un ampio ventaglio di device disponibili attualmente o che verranno realizzati in futuro.

5.2 Le librerie PJSIP

Con il termine "Librerie PJSIP" (18) si intende un insieme di librerie strettamente connesse in grado di essere utilizzate per lo sviluppo di applicazioni basate sullo standard SIP/RTP.
Tali librerie sono state scelte per l'implementazione di un architettura ABPS lato client. Nello specifico per un dispositivo mobile dotato di Symbian OS.

La scelta di adoperare tali librerie per l'implementazione lato client è ricaduta grazie ad una serie di vantaggi:

- **Portabilità**. Le librerie in generale supportano un insieme svariato di piattaforme:

Windows, Linux, Windows Mobile e principalmente Symbian OS.

- **Footprint limitato**. La capacità di occupare poco "spazio" (si parla dell'ordine di pochi KB) lo rende un sistema adatto per apparecchi quali smartphone.
- **High performance**. Vantaggio sempre a favore dell'utilizzo del sistema su smartphone, dato dalla poca occupazione di risorse in quanto a CPU.
- **Ampio set di Features**. quali la gestione della sottoscrizione di eventi, dello stato di presenza, l'instant messaging, il trasferimento di chiamata, ecc...

In realtà utilizzare la terminologia "Librerie PJSIP" non è del tutto corretto, in quanto PJSIP rappresenti appunto solo una delle librerie connesse(figura 5.1) componenti uno stack di librerie. In particolare l'insieme è composto da:

- PJSIP: rappresenta uno stack SIP che supporta un insieme di features ed estensioni del protocollo SIP.
- PJLIB: rappresentante la libreria a cui le restanti si appoggiano. Si occupa di garantire funzionalità di base (es. L'astrazione rispetto al sistema operativo sottostante). Può essere considerata una replica della libreria libc con l'aggiunta di alcune features come la gestione dei socket, funzionalità di logging, gestione thread, mutua esclusione, semafori, critical section, funzioni di timing, gestione eccezzioni e definizione di strutture dati di base (liste, stringhe, tabelle, ecc...).
- PJLIB-UTIL: anch'essa come PJLIB è una libreria di appoggio, ma a differenza della prima implementa funzioni più complesse utili per la crittografia come gli algoritmi: SHA1, MD5, HMAC, CRC32. Ed anche funzioni per il parsing e la manipolazione di testi.
- PJMEDIA: libreria il cui scopo è la gestione ed il trasferimento dei dati multimediali
- PJSUA: rappresenta il livello più alto fungendo da wrapper verso le altre librerie. Inoltre agevola notevolmente la scrittura di applicazioni.

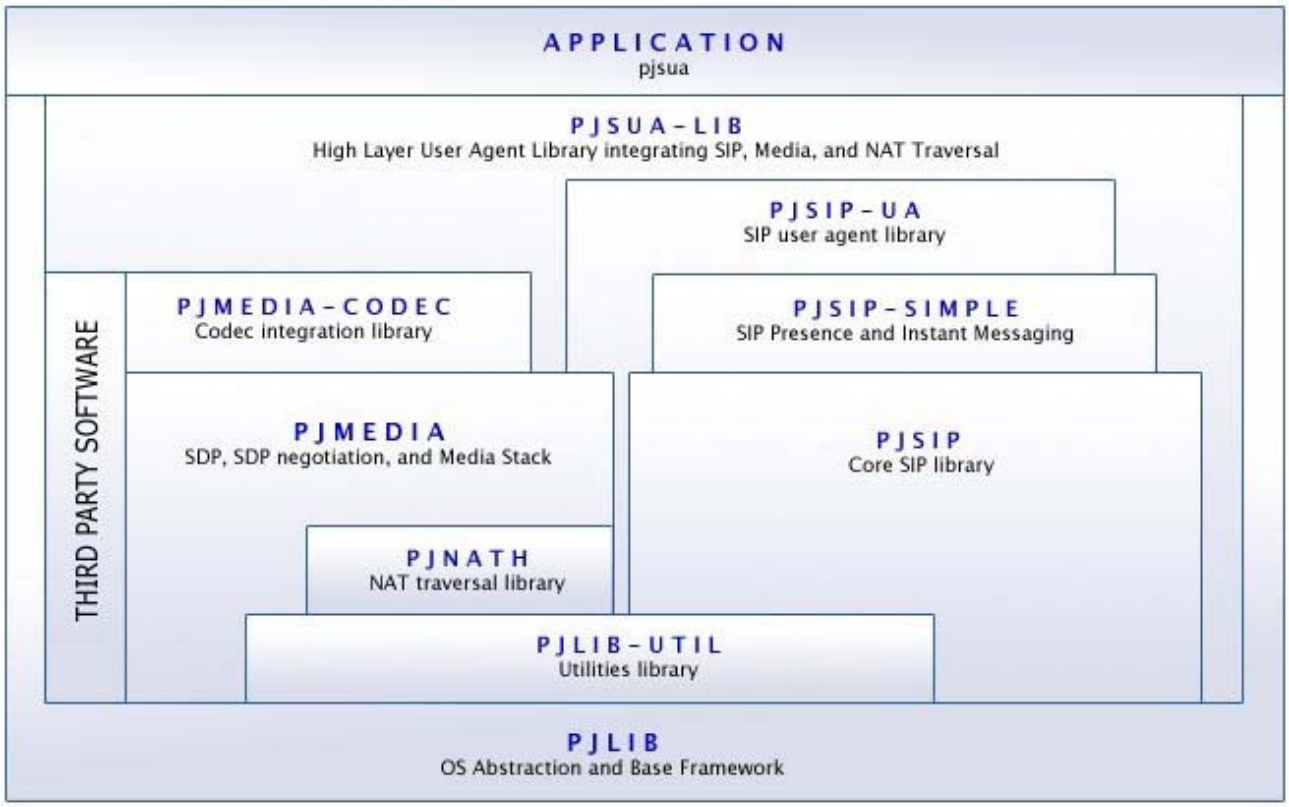

Fig. 5.1 Stack delle librerie PJSIP

5.2.1 Struttura PJSIP

Come precedentemente detto PJSIP è un vero e proprio composto di librerie. In questo paragrafo ci si concentrerà sull'analisi dei metodi di collabarozaione con le altre librerie e della struttura di PJSIP, in quanto rappresenti il fulcro dell'insieme delle librerie. Inoltre in essa sono state apportate le principali modifiche attue a implementare il sistema ABPS su Symbian OS.

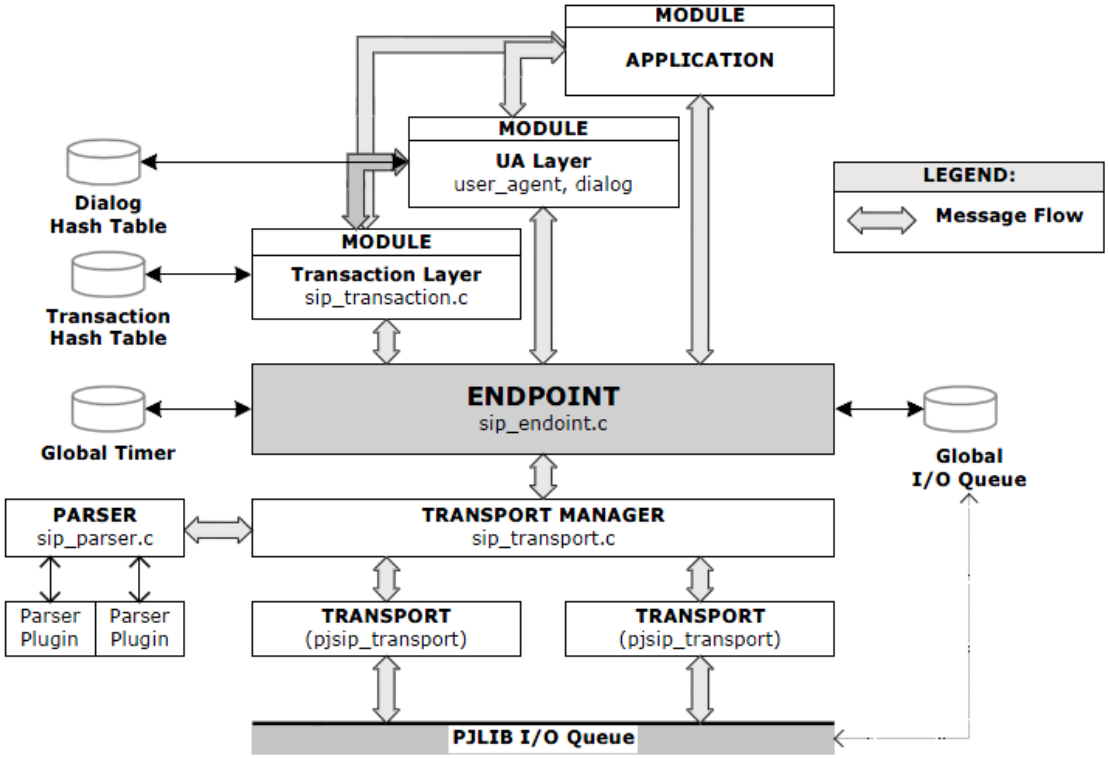

Fig. 5.2 Diagramma di collaborazione PJSIP

Le principali componenti di PJSIP sono:

Endpoint: Rappresentante il cuore dello stack ha principalmente il ruolo di gestire la "pool factory" allocando memoria per tutti i componenti SIP, agisce da scheduler per i vari componenti SIP, gestisce la varie istanze del transport manager il quale si occupa dell'instradamento dei messaggi, detiene una singola istanza della I/O queue, gestisce i moduli PJSIP ed infine riceve i messaggi entranti dal transport manager e si occupa di ridistribuirli ai moduli soprastanti.

Transport: I transport, come suggerisce il nome, hanno il principale scopo di invio e ricezione dei messagi provenienti dalla rete. In generale è possibile dire che il livello di transport ha come fulcro il Transport Manager. Il cui scopo è gestire la creazione di tutti i transport necessari oltre a offrire servizi possibili, quali: instrada i pacchetti provenienti dai vari transport per poi passarli all'endpoint, si occupa di trovare il transport adeguato per l'invio dei messaggi in base al tipo di messaggio da spedire e all'indirizzo remoto, gestisce le

factories dei trasporti e gestisce direttamente la vita dei transport basandosi su un sistema di conteggio delle reference ed un timer di inattività. La libreria pjsip alloca un solo transport manager per endpoint. Il transport manager normalmente non risulta visibile alle applicazioni che devono utilizzare le funzioni esposte dall'endpoint.

Transaction. Il compito di questo "livello" è quello di attuare le giuste fasi di transazione in base ai messaggi SIP in entrata e/o uscita: possibile ritrasmissione messaggi di INVITE o REGISTER in mancanza di una risposta dei primi, risposte in base ai messaggi ricevuti dal server (come un 200 OK o un 503 Service Unavailable). Oltre ad avere delle API necessarie a comunicare all'endpoint i messaggi in uscita e delle funzioni di callback necessarie al monitoraggio delle trasmissioni.

UA. Il concetto astratto dato dallo User Agent (in particolare UA introduce sia la classe user_agent generica sia la classe dialog) è quello della creazione, distruzione ed identificare delle sessioni (INVITE, REGISTER, SUBSCRIBE/NOTIFY, ecc...) necessarie ad implementare correttamente le fasi di una comunicazione SIP.

DataBuffer. ogni messaggio ricevuto viene passato attraverso i vari componenti software incapsulato in una struttura, anziché in come messaggio semplice, questa struttura contiene informazioni aggiuntive che riguardano il messaggio come ad esempio l'istante temporale di ricezione, o l'indirizzo ip del mittente del messaggio stesso. La dichiarazione dei buffer è presente nel file *pjsip/sip_transport.h*, in tale file viene descritto anche il transmit data buffer che è il buffer che viene usato per l'invio dei messaggi.

I/O Queue: Anche se non direttamente appartenenti alla struttura di PJLIB (I/O Queue è un modulo appartenente a PJLIB) è importante citare questo insieme di metodi e funzioni in quanto comunichino direttamente con lo strato Transport. I/O Queue ha il principale compito di fornire un insieme di API per la gestione delle principali operazioni asincrone di Input Output. In particolare essa può lavorare sia su socket sia su descrittori di files, lavorando in maniera nativa in sistemi che, come Symbian OS, gestiscono correttamente le operazioni asincrone. In casi differenti, invece, si limita ad effettuare un polling per simularne il funzionamento.

Callback: PJSIP prevede un meccanismo di callback che hanno origine nel transport manager, che effettua il parsing del messaggio, transitano dell'endpoint e poi vengono propagate da esso verso le altri parti del sistema.

Oltre ai vari componenti dello stack PJSIP. Tale librerie si appoggia sull'utilizzo di un modello framework dei moduli ed una gestione degli header.

Il framework dei moduli è il principale mezzo per distribuire i messaggi SIP tra le componenti software in una applicazione PJSIP, e nelle librerie stesse. Esso è basato su un semplice ma potente meccanismo: per i messaggi in ingresso, l'endpoint distribuisce il messaggio a tutti i moduli, a partire dal modulo con la priorità più alta, finchè ogni modulo non finisce di processare il messaggio. Per i messaggi in uscita, l'endpoint distribuisce i messaggi ai moduli, sempre seguendo l'ordine per priorità, prima che questi vengano trasmessi sull'interfaccia di rete. In questo modo i moduli possono ognuno aggiungere delle modifiche al messaggio, oppure registrarlo per scopi di logging.

```
struct pjsip_module
{
   PJ_DECL_LIST_MEMBER(struct pjsip_module);              // For internal list mgmt.
   pj_str_t             name;                             // Module name.
   int                  id;                               // Module ID, set by endpt
   int                  priority;                         // Priority

   pj_status_t (*load)          (pjsip_endpoint *endpt);  // Called to load the mod.
   pj_status_t (*start)         (void);                   // Called to start.
   pj_status_t (*stop)          (void);                   // Called top stop.
   pj_status_t (*unload)        (void);                   // Called before unload
   pj_bool_t   (*on_rx_request) (pjsip_rx_data *rdata);   // Called on rx request
   pj_bool_t   (*on_rx_response)(pjsip_rx_data *rdata);   // Called on rx response
   pj_status_t (*on_tx_request) (pjsip_tx_data *tdata);   // Called on tx request
   pj_status_t (*on_tx_response)(pjsip_tx_data *tdata);   // Called on tx request
   void        (*on_tsx_state)  (pjsip_transaction *tsx,  // Called on transaction
                                 pjsip_event *event);     // state changed
};
```

Fig. 5.3 Dichiarazione struttura generico modulo PJSIP

Le componenti principali di una struttura che descrive un modulo sono rappresentati da un insieme di dati che ne descrivono il nome, l'id e la sua priorità oltre ad alcuni puntatori a funzione (figura 5.3):

- *on_rx_request(), on_rx_response()*. Hanno il compito di ricevere i messaggi provenienti dall'endpoint o da altri moduli.
- *on_tx_request(), on_rx_response()*. Vengono chiamate dal Transport Manager prima che un messaggio venga trasmesso.
- *on_tsx_state()*. Utilizzata per ricevere notifiche che cambiano lo stato (vedi transaction) in caso di arrivo o invio messaggio, eventi del timer o eventi legati ad errori.

In PJSIP gli header dei messaggi vengono memorizzati in strutture, e condividono tutti delle proprietà comuni come il titolo dell’header, il tipo, un nome abbreviato e una virtual function table. Questo fa sì che lo stack di librerie PJSIP tratti tutti gli header uniformemente.

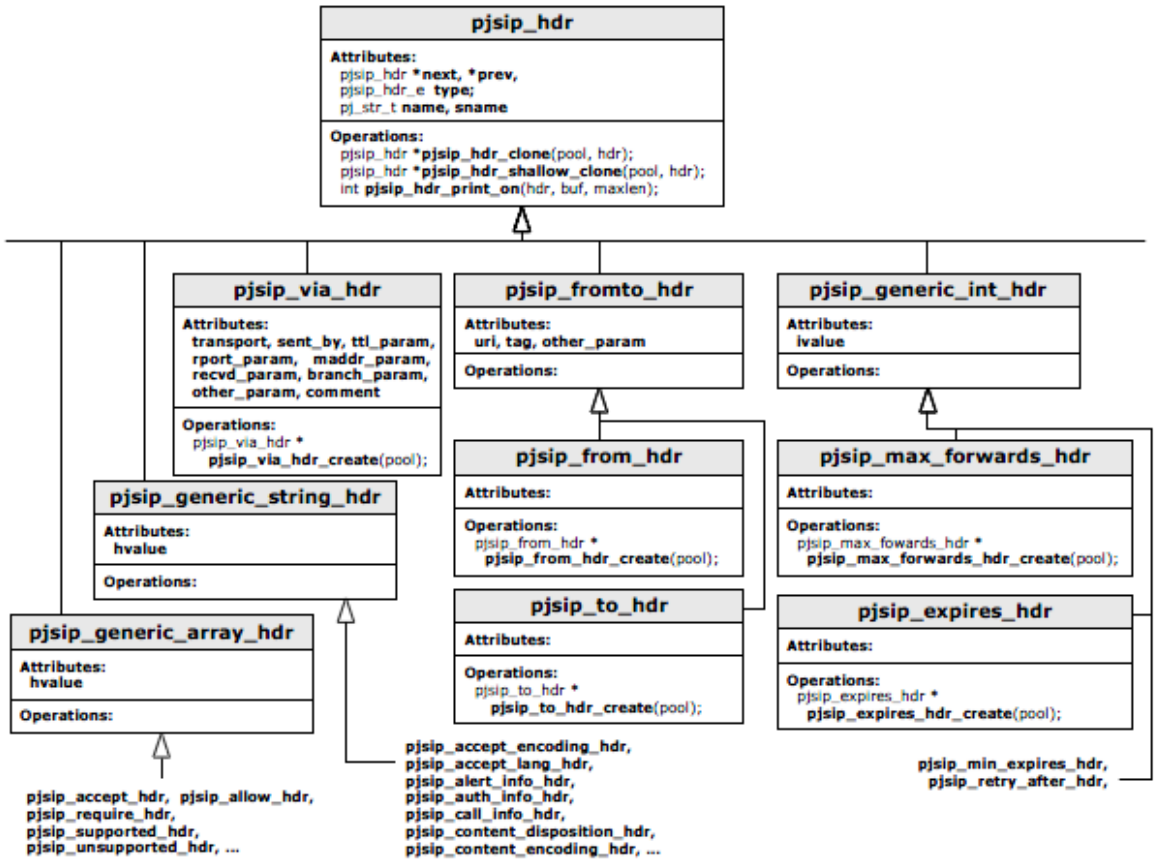

Fig. 5.4 Struttura header PJSIP

L'organizzazione flessibile di PJSIP permette di creare degli header su misura, definendo oltre alla struttura dei campi anche un insieme di funzioni da associare alla *virtual function table* dell'header. In particolare sarà necessario creare una funzione specifica per la struttura dell'header per le operazioni di "clone" (copia dell'intera struttura in una nuova allocazione), "shallow clone" (una copia non completa che conserva puntatori alla struttura sorgente), e "print on" (stampa in memoria dell'header completo). Per facilitare l'implementazione di ABPS su PJSIP sono stati creati gli header di fingerprint, *UID* e *SeqNUM* e le relative funzioni associate alla *virtual function table* di ognuno.

5.3 Struttura per ABPS in PJSIP

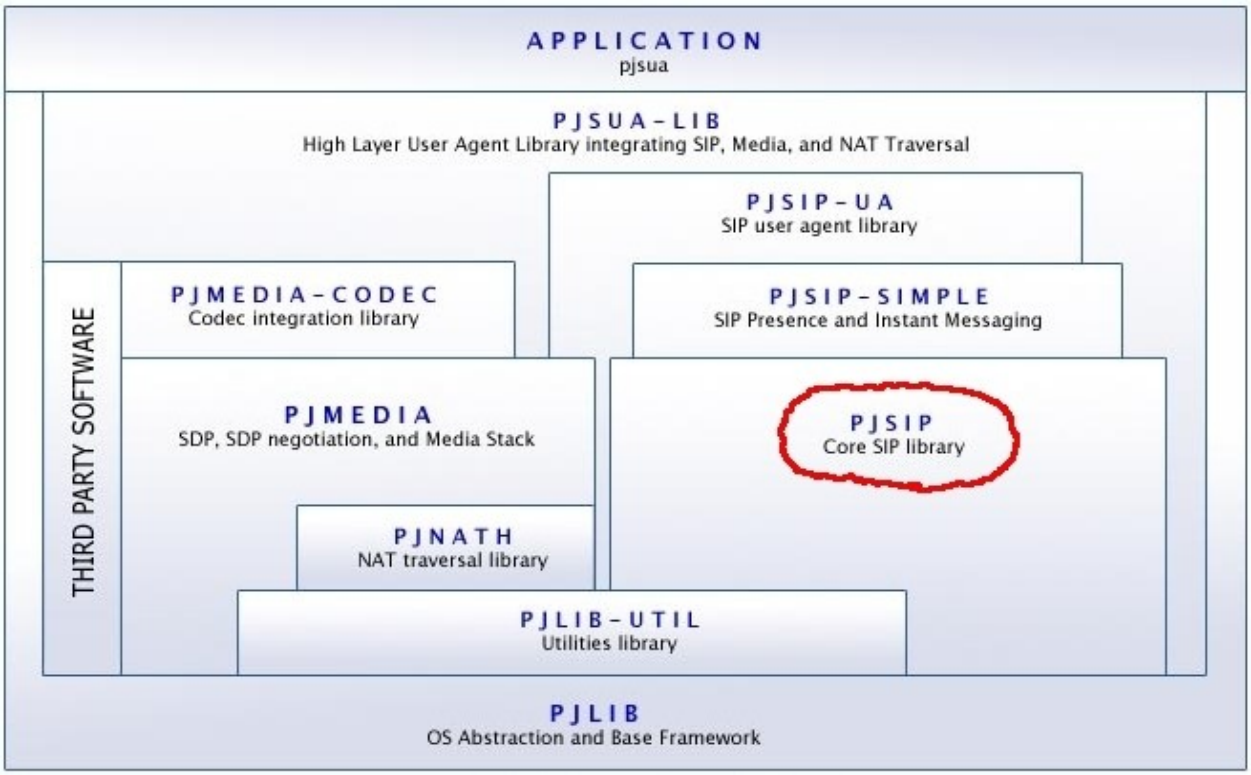

Fig. 5.5 Parte dello stack di librerie PJSIP soggetta a modifiche per l'implementazione di un sistema ABPS

In figura 5.5 è stata cerchiata la libreria su cui principalmente si sono apportate le modifiche necessarie all'estensione di PJSIP in maniera da garantire un autenticazione secondo ABPS. In particolare si parlerà di una aggiunta, più che modifica, di alcune credenziali per la creazione di un account ABPS nel livello più alto (application) ed alcune modifiche inerenti ad una parte dei moduli, responsabili della gestione dei messaggi in entrata e uscita (user_gent, dialog e transaction).

Il processo di autenticazione in PJSIP avviene in diversi passi. Tutto parte dalla configurazione di un account *pjsua*, che avviene nel client, ovvero a livello *Application/pjsua* dello stack PJSIP. La struttura *pjsua_acc_config* memorizza tutte le

informazioni relative ad un account, incluse le credenziali dell'account in una relativa sottostruttura. Successivamente alla configurazione, sempre nello strato *pjsua*, viene richiamata la procedura per rendere attivo l'account, che oltre ad allocare in memoria le strutture create ed inserirle in un vettore di account, unico per l'istanza dello user agent, richiama le procedure per la registrazione al server SIP specificato. Questo passo avviene esclusivamente nel caso in cui l'account richieda la registrazione al server.

Dopo aver creato il messaggio con l'header REGISTER (senza credenziali) da inviare al server SIP in *pjsua_acc_set_registration*, PJSIP si appresta a consegnare il messaggio all'endpoint, passando come argomento una funzione di callback che sarà necessaria successivamente. A questa fase, che rappresenta la prima fase del *four-way handshake* dell'autenticazione HTTP digest, segue la risposta del server SIP ABPS, che ci comunica le credenziali necessarie per la registrazione con un messaggio di errore 407 (in quanto si tratta di un proxy). E' qui che diventa chiara l'utilità della funzione di callback, passata a *pjsip_endpt_send_request* per gestire la risposta del server alla richiesta REGISTER inviata. La funzione di callback *tsx_callback* gestisce molteplici eventi corrispondenti alle diverse risposte del server SIP. Nel nostro caso, di *status code* 407, la funzione di callback richiamerà la procedura per re-inizializzare il messaggio REGISTER (*pjsip_auth_clt_reinit*), questa volta con le credenziali richieste dal server. Quest'ultima procedura, per re-inizializzare il messaggio, riutilizza il primo messaggio inviato al server, ovvero il REGISTER senza credenziali di autenticazione.

Per implementare l'autenticazione dei messaggi secondo le modalità ABPS, sono state create due funzioni per costruire ed aggiungere in coda l'header di autenticazione ABPS, formato da *UID, SeqNUM* e *Fingerprint*, e per verificare gli header dei messaggi ricevuti. Le due funzioni sono richiamate da un modulo pjsip, creato per ABPS, caricato successivamente all'autenticazione mutuale. La prima funzione estrae l'identificativo utente dalla struttura di credenziali della configurazione di connessione ABPS (*pjsua_acc_config*), e il sequence number dalla sessione PJSIP corrente. Il fingerprint viene calcolato sul *pjsip_msg* passato come argomento, utilizzando la funzione crittografica HMAC-SHA-1 sul messaggio, con la chiave di sessione per l'integrità, già calcolata. Successivamente viene costruito l'header con i campi appena generati e viene aggiunto al *pjsip_msg* ricevuto. Il messaggio potrà poi essere inviato tramite il server proxy ABPS al terminale di

destinazione. Una funzione simile è stata costruita per verificare l'integrità e la provenienza dei messaggi ricevuti. La differenza sostanziale tra questa funzione e la prima, sta nel fatto che, prima di recuperare e calcolare i tre header, questi vengono rimossi dal messaggio. Una volta calcolato fingerprint e recuperato *UID* e *SeqNUM*, si verifica che questi coincidano con quelli precedentemente rimossi. Per fare in modo che i messaggi, prima di raggiungere il transport layer per quelli in uscita e dopo aver raggiunto l'endpoint per quelli in entrata, venissero modificati o verificati con il nostro header, è stato creato un modulo PJSIP. Il framework dei moduli in PJSIP è il principale mezzo per distribuire i messaggi SIP tra le componenti software in una applicazione PJSIP e nelle librerie stesse. Tutte le componenti PJSIP infatti, compreso il livello che gestisce le transazioni SIP, sono implementate come moduli. Il framework dei moduli è basato su un semplice ma potente meccanismo: per i messaggi in ingresso, l'endpoint distribuisce il messaggio a tutti i moduli, a partire dal modulo con la priorità più alta, finchè ogni modulo non finisce di processare il messaggio. Per i messaggi in uscita, l'endpoint distribuisce i messaggi ai moduli, sempre seguendo l'ordine per priorità, prima che questi vengano trasmessi sull'interfaccia di rete. In questo modo i moduli possono ognuno aggiungere delle modifiche al messaggio,
oppure registrarlo per scopi di logging.
Inoltre per il sistema ABPS, è stato creato un modulo, inizializzato e caricato ad autenticazione mutuale avvenuta, nel quale i puntatori *on_tx_request* e *on_tx_response* puntano entrambi alla funzione per aggiungere l'header, e i puntatori *on_rx_request* e *on_rx_response* puntano alla funzione per verificare l'autenticità del messaggio ricevuto dal server.

5.4 Le librerie LIBZRTP

Per permettere l'integrazione di un protocollo key-agreement ZRTP all'interno di RTP/SRTP (implementate tramite le PJLIB) sono state utilizzate le librerie LIBZRTP (19).
Per semplicità di comprensione ed utilizzo è possibile suddividere a grandi linee la complessa organizzazione delle librerie LIBZRTP in vari "sottogruppi", a seconda del ruolo ricoperto.

1) *Strutture e funzioni di configurazione*. Un insieme di strutture dati e funzioni atte a costruire un profilo iniziale su cui l'intero protocollo ZRTP si appoggerà. A capo di tutto ci sono due fondamentali strutture *zrtp_config_t* e *zrtp_profile_t*. La prima racchiude un insieme di dati fondamentali, tra cui i più importanti sono un id simbolico da associare al client in uso ed un puntatore a funzioni di callback. Tali callback verranno richiamate al momento del bisogno, in particolare hanno il compito di monitorare eventuali eventi (incluse eccezioni) dati da una post-configurazione o traffico dei pacchetti in arrivo e/o in uscita. E' pertanto necessarrio in fase di configurazione, associare le giuste funzioni di callback in maniera da permettere l'applicazione del protocollo ZRTP nella sessione RTP. *zrtp_profile_t* è invece una struttura che rappresenta un profilo da associare ad una specifica sessione ZRTP (per ogni sessione ZRTP può essere associato uno ed un solo specifico profilo). Le componenti principali del profilo sono una serie di preferenze per il settaggio dei parametri di crittografia (calcolo SAS, Hash, Public Key, ecc...). La configurazione di un profilo può avvenire tramite un settaggio specifico determinato dal programmatore oppure è possibile lasciare al sistema il compito di settare i parametri con i giusti criteri.

2) *Strutture e funzioni di inizializzazione*. *zrtp_config_defaults, zrtp_init* e *zrtp_down* sono le principali funzioni responsabili dell'inizializzazione delle librerie ZRTP. *zrtp_config_defaults* ha il compito, in caso in cui il profilo non sia stato configurato dal programmatore, di settare i parametri adeguati di un profilo (ovviamente la funzione necessita di una chiamata esplicita). *zrtp_init* rappresenta il main delle librerie LIBZRTP, inizializza tutti i suoi componenti, i dati globali, compresa la gestione della memoria. *zrtp_down* ha il compito di deallocare tutte le risorse utilizzate da LIBZRTP.

3) *Creazione e configurazione di una sessione ZRTP*. E' Rappresentata da un insieme di funzioni il cui compito è di creare ed allocare memoria per una sessione ZRTP. *zrtp_session_t* è la struttura rappresentante la sessione da inizializzare (compito affidato alla funzione *zrtp_session_init*) mentre *zrtp_stream_t* rappresenta lo stream su cui avverrà la sessione ZRTP. In particolare *zrtp_stream_t* viene associato, nel momento della sua configurazione, alla sessione corrente in uso.

4) *Attaching di uno stream ZRTP e protocollo di inizializzazione*. Sono presenti una serie di funzioni il cui compito è quello di eseguire un attachment (associazione) tra sessione e stream tale da garantire che tutti i parametri settati sul profilo, sullo stream e sulla sessione operino nella stessa fase.

5) *Traffico RTP, SRTP e RTCP*. LIBZRTP di per se non implementa il trasporto dei pacchetti, lasciando tale compito ad eventuali librerie a cui fa supporto (nel nostro caso PJLIB). Per questo le funzioni appartenenti a questo "gruppo" hanno solo il compito di processare i pacchetti in arrivo ed in uscita dal canale RTP assicurando che i parametri crittografici e no vengano rispettati durante la trasmissione.

5.4.1 L'integrazione di LIBZRTP in PJLIB

L'integrazione tra le librerie LIBZRTP e PJLIB non avviene solamente a livello di codice. La parte iniziale e fondamentale da affrontare è la compilazione; permettere che entrambe le librerie vengano compilate insieme è parte essenziale per garantire una piena "collaborazione" tra i due sistemi.

Prima di affrontare l'integrazione delle librerie, è bene aprire una breve parentesi sui tipi di file e compilatori utilizzati dall'ambiente di sviluppo Carbide.c++ per poter eseguire una corretta compilazione.

Carbide.c++ integra tre differenti tipi di compilazione, tramite GCCE, ARMV5 o WINSCW. Di nostro interesse sarà l'ultimo, in quanto progettato per la creazione di eseguibili (ambiente Windows) compatibili sul simulatore S60. Simulatore su cui sono stati eseguiti i principali test. Le componenti di cui deve essere dotato un progetto, necessarie al compilatore per farne scaturire un applicazione, sono:

- *component.mmp*. Ogni componente appartenente al progetto è dotata di questo tipo di file, il cui scopo è quello di descrivere le proprietà che dovranno essere utilizzate per la corretta compilazione di tale componente: argomenti di compilazione, path relativi

a librerie statiche e/o dinamiche, path relativi agli header necessari e ai file contenenti il codice vero e proprio, ecc... Il numero di componenti per progetto varia a seconda del progetto stesso; per esempio PJLIB è dotata di molteplici componenti e quindi molteplici file *.mmp*, al contrario LIBZRTP si appoggia sul un solo ed unico componente.

– *bld.inf.* Perchè possa avvenire un corretto processo di sviluppo indipendente da un particolare IDE, ogni progetto importato su Symbian dovrà essere dotato di un file descrittivo *bld.inf*. Il corpo di tale file è costituito dalla lista di tutti i *component.mmp* appartenenti al progetto che dovranno essere compilati. Una mancata segnalazione all'interno del corpo di *bld.inf* di un file *.mmp* causa una mancata compilazione di tale componente.

– *abld.bat.* Un file batch-script windows che rappresenta l'output della compilazione e succesivo input per la generazione dell'eseguibile.

Per quanto riguarda LIBZRTP, l'idea di base è stata quella di integrare la directory contenente i relativi file, non come un progetto a parte in grado di "collaborare" con il progetto PJLIB, bensì come una sua componente (figura 5.6). Per permettere ciò si è apportata una piccola modifica al file *bld.inf* inserendo in esso il path relativo al *component.mmp* di libzrtp (nello specifico *libzrtp.mmp*).

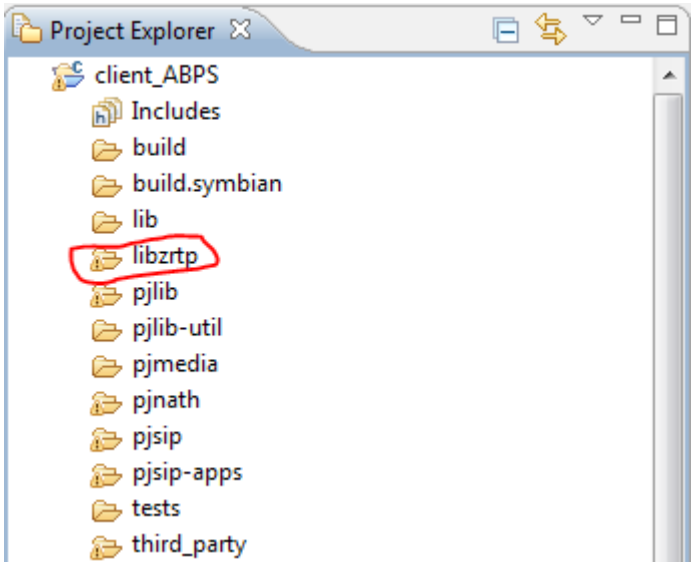

Fig. 5.6 Libzrtp incluso come componente di pjlib.

Le modifiche del file *bld.inf* non sono sufficienti a garantire un corretto funzionamento dei moduli offerti da libzrtp all'interno di pjlib. Occorre aggiungere all'interno dei file *.mmp*, appartenenti ai componenti pjlib che fanno uso delle librerie zrtp, i path relativi ai file header di libzrtp. In particolare i *component.mmp* modificati sono:

- *symbian_ua.mmp*. In quanto la componente di PJLIB legata a tale file rappresenta il main dell'applicazione dove avvengono anche le dichiarazioni delle strutture necessarie a libzrtp.
- *pjsua_lib.mmp*. La componente User Agent di livello più alto. LIBZRTP viene utilizzata per applicare le proprie modifiche prima e durante l'instaurazione di un canale RTP/SRTP.

Una volta integrata, la componente LIBZRTP richiede la sua inizializzazione, il caricamento dei suoi moduli e delle strutture dati in maniera separata da PJLIB. In particolare sono stati presi in considerazione due possibili differenti procedure:

1) Caricamento LIBZRTP in fase post-autenticazione. LIBZRTP viene caricato immediatamente dopo la fase, avvenuta con successo, di autenticazione.
2) Caricamento LIBZRTP in fase pre-chiamata. LIBZRTP verrà caricata solamente a richiesta dell'utente, ovvero dal momento in cui l'utente decide di affettuare o ricevere una chiamata selezionando la rispettiva voce dal menu di PJLIB.

Di per se non esiste un modo migliore tra i due elencati. Il primo garantisce che il caricamento avvenga subito, eliminando la problematica di attesa da parte dell'utente in fase pre-chiamata/recezione ma, a suo contro, richiede della memoria che non necessariamente verrà usata (l'utente può non voler effettuare una chiamata ma eseguire altre operazioni, ad esempio offline). Il secondo elimina la questione della memoria, occupandola solamente quando serve, ma introduce il problema del tempo di attesa da parte dell'utente in fase di chiamata o recezione di una chiamata, dato che per ora il caricamento di LIBZRTP richeide un tempo di più o meno di 1-2 secondi.

5.5 Le problematiche in fase di implementazione

Le problematiche in fase di implementazione del sistema ABPS sono state molteplici ed in gran parte risolte, sia lato client per quanto riguarda PJLIB, sia lato server per quanto riguarda l'applicazione proxy server utilizzata per la gestione delle fasi di autenticazione in ABPS, siproxd.

La prima problematica che si è dovuto affrontare è stata una non corretta, ed in alcuni casi nulla, autenticazione iniziale durante la fase di REGISTER SIP. Questa problematica ricopriva entrambe le autenticazioni necessarie, verso ABPS e verso il servizio VoIP a cui l'utente è registrato (in fase di test è stato utilizzata un identità ekiga).

I fattori individuati e corretti, responsabili della non corretta autenticazione sono principalmente tre:

1) Le fasi di autenticazione non avvenivano in maniera ordinata (lato client). Il sistema richiede che avvenga prima l'autenticazione ABPS e successivamente quella verso il servizio VoIP. Nella maggior parte del test le autenticazioni venivano invertite.
2) Al pacchetto SIP di REGISTER inviato in risposta al 407 ricevuto dal server veniva aggiunto, nel campo *content-lenght,* un carattere "bianco" in eccesso causando in fase di controllo del fingerprint, da parte del server stesso, un errore di fingerprint mismatch.
3) Un presunto firewall nella rete (ancora non si è ben chiaro, probabilmente se si tratta della rete Unibo) modificava il campo Contact del pacchetto rivevuto dal server, dove l'indirizzo IP del mittente veniva sostituito con l'indirizzo IP del modem/router che instrada il pacchetto lato client. Di conseguenza anche qui avveniva un errato calcolo del fingerprint (fingerprint mismatch).

Per quanto riguarda il primo problema la soluzione adottata, unicamente client, è stata quella di inserire un controllo che permettesse alle autenticazioni di avvenire in maniera corretta. Di preciso all'interno delle strutture *pjsua_acc* e *pjsua_acc_config,* necessarie a descrivere la struttura degli account in uso, è stata inserita una variabile *auth_abps* con il

compito di identificare se l'autenticazione ABPS sia avvenuta con successo. Infatti solo se l'autenticazione ABPS è avvenuta con successo la variabile (trattasi di un valore booleano) viene settata a TRUE con la possibilità all'account di eseguire l'autenticazione verso il servizio VoIP. In caso contrario l'account non può essere utilizzato per effettuare nessun tipo di registrazione al di fuori di ABPS. Il compito di cambiare il valore alla variabile *auth_abps* è lasciato alla funzione callback *on_reg_state* che verifica lo stato di registrazione di ogni account presente.

L'apportazione della prima modifica non ha eliminato completamente il problema legato all'autenticazione, infatti durante le varie fasi di test capitava che una buona percentuale di autenticazioni non andassero a buon fine. Tramite una fase di controllo lato server si è presto capito che il problema riguardava anche l'invio dei pacchetti, ovvero il secondo e terzo problema precedentemente descritti. Per la risoluzione del secondo problema è stato implementato un meccanismo lato client il cui compito è quello effettuare un controllo del pacchetto che verrà spedito (la funzione viene applicata a tutti i pacchetti SIP che verranno inviati e non solo i REGISTER), identificare la presenza del carattere "bianco" in più e, se presente, eliminarlo. Tale modifica è stata apportata nelle funzioni *process_auth* e *auth_on_tx_abps,* il cui rispettivo ruolo è quello di effettuare il challenge in base alla risposta ottenuta dal server e aggiungere i parametri di autenticità ai messaggi uscenti dal client (inserimento dei parametri richiesti da ABPS: *UID, SeqNUM* e *Fingerprint*). Il terzo problema ha richiesto una modifica sia lato client sia lato server. Ad ogni messaggio SIP inviato (sia server sia client) veniva modificato l'indirizzo IP del campo contact da un firewall presente nella rete che opera sui pacchetti SIP. La soluzione adottata non è definitiva, può considerarsi più una cosa momentanea in attesa di trovare una soluzione più efficiente. Trattasi infatti di un metodo implementato il cui scopo è quello di ingannare semplicemente il firewall responsabile della modifica al campo *Contact.* Ad ogni pacchetto SIP uscente viene modificato il corpo dell'attributo *Via*, ovvero, la stringa "SIP" sostituita con una stringa "VVV". Tutto ciò garantisce che il pacchetto inviato non venga identificato come SIP di conseguenza non gli verranno applicate modifiche da parte del firewall. Come già detto, la risoluzione di tale problema ha richiesta la modifica a livello di codice sia su PJLIB sia sul proxy siproxd tramite l'implementazione di una semplice funzione. Ogni pacchetto SIP, dopo la sua formazione, viene settato dei parametri di autenticità nella solita

funzione *auth_on_tx_abps* ed infatti è qui che viene richiamata la funzione di modifica attributo *Via.* Inoltre dato che questa funzione è stata implementata anche su siproxd, è logico pensare che sarà necessario un altra funzione il cui compito è fare l'operazione inversa (sostituzione da "VVV" a "SIP"), in quanto la mancata operazione inversa creerebbe dei problemi nel riconoscimento del messaggio da parte delle applicazioni in uso. Per ciò tale processo inverso (lato client) è garantito da un altra funzione richiamata all'interno della funzione attua a verificare i parametri dei pacchetti ricevuti, ovvero, *pjsip_tpmgr_receive_packet.*

Un ultima problematica riscontrata, riguarda la fase di scambio di pacchetti SIP di INVITE, facenti parte della pre-fase prima dell'instaurazione di una sessione RTP. Il tutto nasce dal fatto che ogni pacchetto ricevuto da siproxd viene prima scompattato da una serie di funzioni e solo successivamente ricomposto ed utilizzato per verificare i parametri di autenticità richiesti dal sistema. Nello specifico, il messaggio SIP di INVITE veniva ricomposto con una struttura differente dall'originale (la differenza sta negli attributi SDP Allow e Supported) e questo creava il solito problema di fingerprint mismatch ed il conseguente fallimento dell'autenticazione. La soluzione apportata (lato server) è stata quella di invertire le fasi, effettuando prima i controlli di autenticazione e successivamente lo scompattamento del pacchetto. Questa problematica è comunque lasciata ancora aperta, la soluzione adottata è solo momentanea. Infatti bisogna ancora verificare qual'é il giusto standard da utilizzare: quello dell'INVITE inviato (PJSIP) oppure quello del pacchetto, una volta ricevuto e scompattato, ricomposto da Siproxd.

Capitolo 6

Conclusioni

Le tencologie VoIP, i dispositivi mobile, il multihoming le sue problematiche fin qua spiegate sono un insieme di tecnologie in continua evoluzione, oramai facenti parte di un servizio a disposizione di tutti. Il far si che queste innovazioni possano interagire con minor problemi possibili e garantire le migliori prestazioni possibili, è quindi un obiettivo molto importante da raggiungere per tecnologia informatica. Il sistema ABPS, implementato tramite l'utilizzo di un framework PJSIP ed un protocollo di key-agreement, garantito dalla libreria LIBZRTP, è un ottima soluzione alle problematiche di samless mobility e multihoming, in grado di unire fortemente concetti teorici a concetti pratici. Ed è per questo che si necessita di un forte e continuo sviluppo di tale sistema, che in futuro, sarà anche in grado di essere esteso su piattaforme differenti da quella di base utilizzata (Symbian). Entrando nello specifico, sull'implementazione del sistema ABPS, si è arrivati ad una fase in cui la parte di autenticazione risulta funzionante; ogni bug è stato corretto ed il processo migliorato. Rimane ancora aperta la questione legata all'instaurazione di una sessione RTP, con lo scopo di eseguire una fase di comunicazione audio. Il protocollo di key-agreement ZRTP (LIBZRTP) è stato integrato pienamente in PJLIB e la fase di calcolo ed inserimento dell'hello-hash zrtp nell'SDP del messaggio di INVITE SIP effettuato con successo. Rimanenti sono le fasi successive, sarà infatti necessario studiare a fondo la giusta tecnica di implementazione. Specialmente per quanto riguarda il transporter RTP da utilizzare, in quanto, un accurato controllo, ha fatto emergere il vantaggio di implementare un transport a parte senza estendere quello preesistente offerto da PJSIP.

Bibliografia

1. **International Engineering Consortium.** "*Voice over Internet Protocol. Definition and Overview". 2007.*
2. ***Information Sciences Institute,* University of Southern California.** *"Internet Protocol RCF 791". 1981.*
3. **Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler.** "*SIP: Session Initiation Protocol RFC 3261 ".* 2002.
4. **H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson.** *"RTP: A Transport for Real-Time Applications RFC 3550 (Proposed Standard)".* 2003.
5. **T. Friedman, Ed. Paris 6 R. Caceres, Ed. IBM Research A. Clark, Ed. Telchemy.** *"RTP Control Protocol Extended Reports (RTCP XR)".* 2003.
6. **M. Baugher, D. McGrew, Cisco Systems, Inc., M. Naslund, E. Carrara, K. Norrman, Ericsson Research.** *"The Secure Real-time Transport Protocol (SRTP) RFC 3711".* 2004.
7. **J. Postel.** *"UDP: User Datagram Protocol RCF 768".* 1980.
8. **wi-fi.org.** *Wi-Fi Alliance [online].*
9. **Niclas Ek, Department of Electrical Engineering. Helsinki University of Technology.** "*IEEE 802.1 P,Q - QoS on the MAC level".* 1999.
10. **R. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc.** *"The MD5 Message-Digest Algorithm RCF 1321". 1992.*
11. **D. Eastlake, 3rd, Motorola, P. Jones,Cisco Systems.** *"US Secure Hash Algorithm 1 (SHA1) RCF 3174". 2001.*
12. **H. Krawczyk, IBM, M. Bellare, UCSD, R. Canetti.** *"HMAC: Keyed-Hashing for Message Authentication RCF 2104 ". 1997.*

13. **J. Franks Northwestern University, P. Hallam-Baker Verisign, Inc. J. Hostetler AbiSource, Inc. S. Lawrence Agranat Systems, Inc. P. Leach Microsoft Corporation A. Luotonen Netscape Communications Corporation L. Stewart Open Market, Inc.** *"HTTP Authentication: Basic and Digest Access Authentication RCF 2617". 1999.*

14. **E. Rescorla RTFM Inc.** *"Diffie-Hellman Key Agreement Method RCF 2631". 1999.*

15. **P. Zimmermann Zfone Project A. Johnston, Ed. Avaya J. Callas Apple, Inc.** *"ZRTP: Media Path Key Agreement for Unicast Secure RTP draft-zimmermann-avt-zrtp-22".* 2010.

16. **Vittorio Ghini et al.** *"Always Best Packet Switching" for SIP-based mobile multimedia services. 2009.*

17. **Symbian.org.** *Developer Wiki [online].*

18. **pjsip.org.** *PJSIP Developer's Guide Version 5.0.4 [online].*

19. **zfone.com.** *Libzrtp Detail Description API [online].*